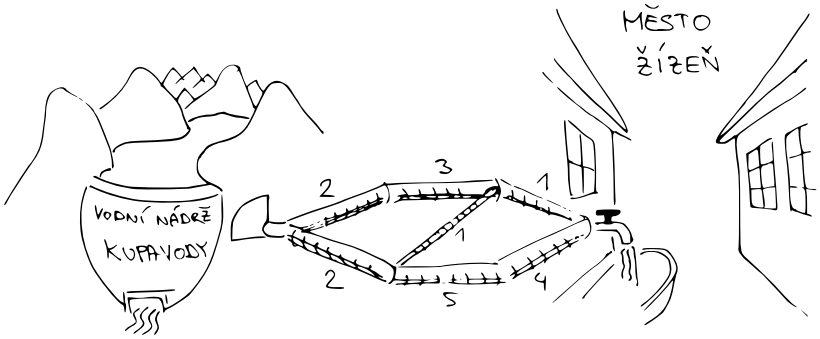
Kapitola 1

Toky v sítích



Motivace (pro toky): Ve městě Žížeň je velký nedostatek vody. Město je propojeno potrubní sítí s vodní nádrží Kupavody, kde je vody dostatek. Schéma vodovodní sítě je na obrázku. Každá trubka je jinak široká a proto je ve schématu u každé trubky napsán maximální počet litrů, který trubkou proteče za jednu minutu. Vaším úkolem je zjistit, kolik nejvíce litrů doteče z přehrady do města.[1]

Toky v sítích nejsou jen o vodě v potrubí. V analogických sítích může protékat elektrický proud, auta ve městě, telefonní hovory, peníze nebo pakety v počítačových sítích.[2]

Motivace (pro řezy): V Hádavém království jsou některé dvojice měst spojeny přímou silnicí. Po silniční síti se dá dostat z každého města do libovolného jiného. Jak už to tam bývá zvykem, dvě velká města Velezdroje a Hustostoky se pohádala o to, z které strany se loupe banán. Radní obou měst chtějí rozkopat některé silnice tak, aby už nebylo možné dojet po silnici z jednoho města do druhého. Prý tím zabrání šíření špatných názorů. Které silnice mají silničáři překopat, aby splnili úkol a zároveň překopali co nejméně silnic?

[1] Je zajímavé, že do skutečného potrubí stačí vodu pustit a ona už sama poteče tak, aby jí proteklo co nejvíce.

[2] Problém toků v sítích zkoumali už v 50. letech výzkumníci Air Force T. E. Harris a F. S. Ross. Studovali přesun materiálu po železniční síti ze Sovětského Svazu do satelitních zemí východní Evropy. Minimální řezy železniční sítě jsou strategická místa pro americké bombardéry. Jejich výzkum byl a přísně tajný a odtajněn byl až v roce 1999.

Protože je kapitola o tocích v sítích poměrně rozsáhlá, pojďme si stručně představit obsah hlavních sekcí.

- 1.1 Maximální tok a minimální řez – úvodní kapitola, ve které zavedeme důležité pojmy a vysvětlíme základní teorii.
- 1.2 Algoritmy vylepšující cesty – historicky starší algoritmy (Ford-Fulkersonův, Dinicův, Edmons-Karpův, 3 Indů), které pro nalezení maximálního toku využívají teorii o vylepšujících cestách.
- 1.3 Goldbergův Push-Relabel algoritmus – novější a prakticky rychlejší algoritmus, který není přímočarou aplikací teorie o tocích.
- 1.5 Aplikace toků v sítích – ukázka několika problémů, které se dají vyřešit převodem na hledání maximálního toku. Toky v sítích mají ohromné množství aplikací. S dalšími aplikacemi se seznámíme ve cvičeních na konci kapitoly.

1.1 Maximální tok a minimální řez

Definice: *Síť* (G, s, t, c) je orientovaný graf $G = (V, E)$, který má dva speciální vrcholy: *zdroj* s a *spotřebič* $t \in V$ (z anglického source a target), a každá hrana e má *kapacitu* $c(e)$, kde kapacita je funkce $c : E \to \mathbb{R}_+$. Spotřebič je někdy označován jako *stok*.

Tok f je funkce $f : E \to \mathbb{R}_+$, která splňuje

$$\begin{array}{lll} i) & 0 \le f(e) \le c(e) & \text{pro každou hranu } e \in E \\ ii) & \sum_{xv \in E} f(xv) = \sum_{vx \in E} f(vx) & \text{pro každý vrchol } v \in V \setminus \{s,t\} \end{array}$$

Číslu $f(e)$ říkáme tok po hraně e nebo také průtok hranou e. První podmínka říká, že průtok hranou je nezáporný a nemůže překročit kapacitu hrany. Druhá podmínka říká, že co do vrcholu přiteče, to z něj zase musí odtéci. Druhé podmínce se také říká zákon zachování toku, protože se tok ve vrcholech ani neztrácí[3], ani nepřibývá (kromě zdroje a spotřebiče). V analogii v elektrických obvodech se druhé podmínce říká *Kirchhoffův zákon.* Pokud chceme explicitně vyjádřit, že mluvíme o toku ze zdroje s do spotřebiče t, tak tok označíme jako (s, t)-tok.

Protože se v teorii o tocích vyskytuje rozdíl mezi přítokem do vrcholu v a odtokem z vrcholu v hodně často, tak si zavedeme zkrácené označení

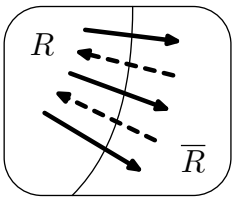
$$f(v) = \sum_{xv \in E} f(xv) - \sum_{vx \in E} f(vx).$$

Číslu $f(v)$ budeme říkat *bilance vrcholu* v nebo také *přebytek toku* ve vrcholu v. Při čtení následujícího textu budeme muset být malinko opatrní na to, co je argumentem $f()$, protože $f(e)$ nebo $f(uv)$ je průtok hranou $e = uv$, ale $f(v)$ je bilance vrcholu v. Druhou podmínku z definice toku můžeme jednoduše vyjádřit jako $f(v) = 0$.

Velikost toku $|f|$ je množství toku, které protéká ze zdroje do spotřebiče. Protože se tok ve vrcholech ani neztrácí ani nepřibývá, tak ho můžeme spočítat jako $|f| = f(t)$. Tedy jako tok, který přitéká do spotřebiče. Stejně bychom mohli velikost toku měřit jako $-f(s)$, což je velikost toku, který vytéká ze zdroje.

Doplněk množiny $R \subseteq V$ značíme $\overline{R}$. Tedy $\overline{R} = V \setminus R$. Množinu orientovaných hran $\delta(R) = \{vw \in E \mid v \in R \,\&\, w \in \overline{\mathcal{R}}\}$ nazveme *řezem* určeným množinou $R \subseteq V$. Řez je (s, t)-*řez,* pokud $s \in R$ a $t \notin R$. Tedy pokud řez odděluje zdroj od spotřebiče. Řez určený jediným vrcholem v budeme zjednodušeně značit $\delta(v)$ místo $\delta(\{v\})$. *Kapacita řezu* $c(\delta(R)) := \sum_{e \in \delta(R)} c(e)$.

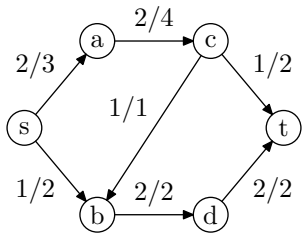
Pozor, je rozdíl mezi řezem v orientovaném grafu a řezem v neorientovaném grafu. Řez určený množinou R v orientovaném grafu obsahuje jen hrany, které z R vychází ven. Na proti tomu řez určený množinou R v neorientovaném grafu obsahuje hrany, které mají jeden konec v R.



Pokud chceme hledat maximální tok nebo minimální řez v neorientovaném grafu, tak nahradíme každou neorientovanou hranu dvojicí šipek jdoucích proti sobě. Kapacity obou šipek budou stejné jako kapacita původní hrany. Maximální tok a minimální řez v takto vytvořeném orientovaném grafu odpovídá toku a řezu v původním neorientovaném grafu. Pro jednoduchost budeme v celé kapitole o tocích pracovat pouze s orientovanými grafy.

[3] To už o pražské vodovodní síti říci nejde. Před lety se uvádělo, že se ztratí až pětina vody, která se do potrubí pustí.

Příklad: Na následujícím obrázku je síť. U každé hrany e jsou uvedeno „$f(e)/c(e)$“.



Velikost toku v síti na obrázku je 3. Velikost řezu určeného vrcholy $\{s, a, b\}$ je 6 a velikost opačného řezu, to je řezu určeného vrcholy $\{c, d, t\}$, je 1.

Platí věta, že v každém grafu existuje maximální tok. Přímý důkaz věty vyžaduje pokročilejší znalosti matematické analýzy a proto ho neuvedeme. Větu dokážeme nepřímo tím, že si ukážeme algoritmy, které maximální tok najdou.

Lemma 1 *Pro každý (s,t)-tok f a každý (s,t)-řez $\delta(R)$ platí*

$$f(\delta(R)) - f(\delta(\overline{R})) = f(t)$$

Důkaz: Spočítáme $X = \sum_{v \in \overline{R}} f(v)$ dvěma způsoby. Jednou přes příspěvky vrcholů, podruhé přes příspěvky hran.

Bilance všech vrcholů kromě zdroje a spotřebiče je rovna nule. Množina $\overline{R}$ obsahuje jen spotřebič t a proto $X = f(t)$.

Na druhou stranu se podívejme na příspěvky od jednotlivých hran. Hrana $e = vw$ s oběma konci v $\overline{R}$ přispívá do bilance vrcholu v hodnotou $-f(e)$ a do bilance vrcholu w hodnotou $f(e)$. Proto je její celkový příspěvek do X roven nule. Jediné hrany, které přispívají do X něčím nenulovým, jsou ty které mají jeden konec v R a druhý v $\overline{R}$. Můžeme je rozdělit na hrany $\delta(R)$ vedoucí z R ven a na hrany $\delta(\overline{R})$ vedoucí z venku do R. Jejich příspěvky do X jsou $f(\delta(R)) - f(\delta(\overline{R}))$. ∎

Důsledek 1 *Pro každý (s,t)-tok f a každý (s,t)-řez $\delta(R)$ platí*

$$f(t) \leq c(\delta(R))$$

Důkaz: Z lemma 1 dostáváme $f(t) \leq f(\delta(R)) \leq c(\delta(R))$. Druhá nerovnost platí, protože kapacita hrany je horním odhadem na průtok hranou. ∎

Velikost každého (s,t)-řezu je horním odhadem na velikost toku. Řez je zároveň jednoduchým a snadno ověřitelným certifikátem, jak dokázat, že v síti větší tok neexistuje. Dokonce platí následující věta, která byla objevena Fordem a Fulkersonem v roce 1956 a nezávisle Kotzigem[4] v témže roce.

Věta 1 (o maximálním toku a minimálním řezu) *Pokud existuje maximální (s,t)-tok, pak*

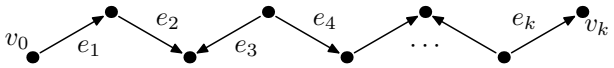
$$\max\{|f| : f \text{ je } (s,t)\text{-tok}\} = \min\{c(\delta(R)) : \delta(R) \text{ je } (s,t)\text{-řez}\}$$

Společně s důkazem věty si ukážeme i základní myšlenku, jak zvětšovat velikost toku. Předpokládejme, že už známe nějaký tok f. Pokud v grafu najdeme orientovanou cestu sPt takovou, že pro každou hranu e cesty P je $f(e) < c(e)$, tak zvětšíme průtok cestou P a tím zvětšíme i tok f. Tato myšlenka ale sama o sobě nestačí. Proč?

[4] Kotzig byl Slovák působící na bratislavské Vysoké škole ekonomické.

Pokud ze spotřebiče do zdroje vede hrana ts s průtokem $f(ts) > 0$, tak celkový tok zvětšíme tím, že z s do t pošleme tok o velikosti $f(ts)$ po hraně ts v protisměru. Ve skutečnosti nic v protisměru nepoteče. Průtoky hranou jdoucí proti sobě se odečtou. Místo toho, aby se z t posílal do s tok o velikosti $f(ts)$ (původní tok po hraně) a zároveň z s do t tok o velikosti $f(ts)$ (přidávaný tok po hraně), tak se vrcholy s a t dohodnou, že si každý nechá tok o velikosti $f(ts)$. Nebudou si nic vyměňovat, protože to vyjde na stejno.

Množství toku, které můžeme poslat po směru hrany e, nazveme *rezerva po směru hrany*. Její velikost je $c(e) - f(e)$. Množství toku, které můžeme poslat proti směru hrany e, nazveme *rezerva proti směru hrany*.[5] Její velikost je $f(e)$.



Cestou v následující definici vylepšující cesty myslíme cestu, u které ignorujeme orientaci hran. Hranu cesty zorientovanou směrem od zdroje do spotřebiče nazveme *dopřednou* a opačně orientovanou hranu nazveme *zpětnou*. Cesta $v_0e_1v_1e_2v_2 \dots e_kv_k$ je *vylepšující cesta pro tok* f, pokud pro každou dopřednou hranu e cesty platí $f(e) < c(e)$ a pro každou zpětnou hranu platí $f(e) > 0$. Cestu vedoucí ze zdroje s do spotřebiče t budeme zkráceně označovat jako (s,t)-cestu.

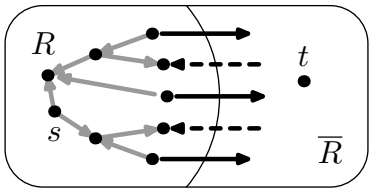
Lemma 2 *Pokud v síti existuje vylepšující cesta P pro tok f vedoucí ze zdroje do spotřebiče, tak tok f není maximální.*

Důkaz: Tok f můžeme vylepšit podél vylepšující cesty sPt. Nechť $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\}$, kde $\varepsilon_1 = \min\{c(e) - f(e) \mid e \in P \text{ je dopředná}\}$ a $\varepsilon_2 = \min\{f(e) \mid e \in P \text{ je zpětná}\}$. Slovy se dá říci, že ε je největší tok, který se dá poslat ze zdroje do spotřebiče podél cesty P. Vylepšením toku f podél cesty P dostaneme tok f'.

$$f'(e) := \begin{cases} f(e) + \varepsilon, & e \in P \text{ je dopředná hrana,} \\ f(e) - \varepsilon, & e \in P \text{ je zpětná hrana,} \\ f(e), & e \notin P. \end{cases}$$

Funkce f' je opět tok, protože splňuje definici toku (ověřte). ∎

Důkaz: (Věty 1 o maximálním toku a minimálním řezu) Každý tok je menší nebo roven velikosti libovolného řezu (důsledek 1). To dokazuje první nerovnost.



Pro důkaz druhé nerovnosti vezmeme maximální tok f a budeme chtít najít řez stejné velikosti. V síti neexistuje vylepšující cesta ze zdroje do spotřebiče, protože jinak se dostaneme do sporu s lemmatem 2.

Nechť $R = \{v \in V$ do kterých vede vylepšující cesta z $s\}$. Množina $\delta(R)$ je (s,t)-řez, protože z s do s vede vylepšující cesta nulové délky, ale z s do t žádná vylepšující cesta nevede. Pro každou hranu řezu $\delta(R)$ je $f(e) = c(e)$ a pro každou hranu řezu $\delta(\overline{R})$ je $f(e) = 0$. Jinak by šla vylepšující cesta prodloužit do vrcholů mimo R. Podle lemmatu 1 je velikost toku $f(s) = f(\delta(R)) - f(\delta(\overline{R})) = c(\delta(R))$, což jsme chtěli ukázat. ∎

Důsledek 2 *Tok f je maximální $\iff$ neexistuje vylepšující cesta pro tok f.*

Předchozí důkaz nám dokonce ukazuje, jak najít minimální řez, který dosvědčí, že větší tok neexistuje.

[5] Později (u Dinicova algoritmu) rozšíříme graf G tak, aby ke každé hraně uv existovala opačná hrana vu, a díky tomu zavedeme rezervy trochu jednodušším způsobem.

1.2 Algoritmy vylepšující cesty

1.2.1 Ford-Fulkersonův algoritmus

Důkaz věty o maximálním toku a minimálním řezu je zároveň i návodem jak hledat maximální tok. Začneme s nulovým tokem a postupně budeme hledat vylepšující cesty, podél kterých zvětšíme aktuální tok. Když už nebude existovat vylepšující cesta, tak máme maximální tok. Dostáváme tak Ford-Fulkersonův algoritmus (z roku 1957).

```
Ford-Fulkerson:
    f := 0
    while existuje vylepšující cesta P z s do t do
        vylepši tok f podél cesty P
    return f
```

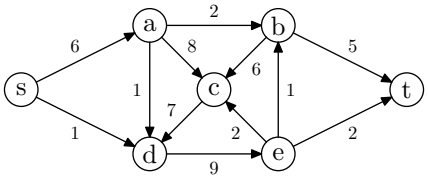
Jak hledat vylepšující cesty si vysvětlíme až v další sekci u Dinicova algoritmu.

Podívejme se, jak je to s konečností Ford-Fulkersonova algoritmu. Pokud má síť celočíselné kapacity, tak v každém kroku stoupne velikost toku alespoň o jedna. Součet všech kapacit, který je horním odhadem na velikost toku, je konečné číslo a proto se algoritmus po konečně mnoha krocích zastaví. Pokud má síť racionální kapacity, tak je můžeme přenásobit nejmenším společným jmenovatelem a tím úlohu převedeme na předchozí případ (průběh algoritmu se tím nezmění). Je zajímavé, že důkaz konečnosti nemůžeme rozšířit na sítě s iracionálními kapacitami. Dokonce existují sítě s iracionálními kapacitami, ve kterých se algoritmus zacyklí (viz třetí z následujících příkladů).

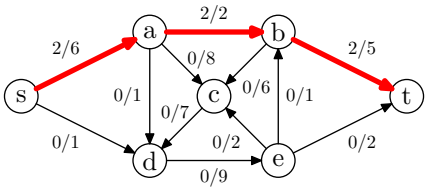
V celočíselné síti zvýšíme tok podél vylepšující cesty vždy o celé číslo. Proto dostáváme následující důsledek.

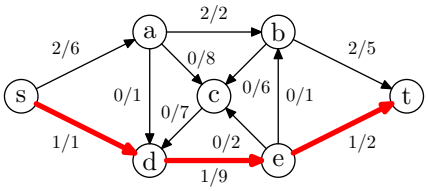
Důsledek 3 *V síti s celočíselnými kapacitami hran existuje maximální tok, který je celočíselný.*

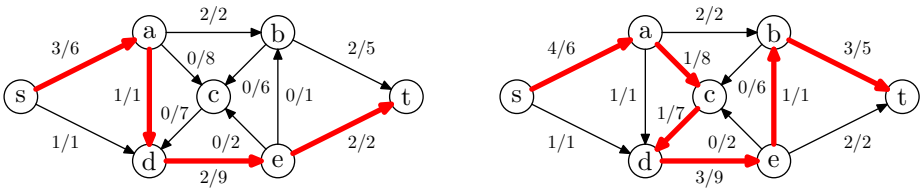
Příklad: (průběh algoritmu) Najděte maximální tok v síti na následujícím obrázku.



Budeme postupovat podle Ford-Fulkersonova algoritmu. Začneme s nulovým tokem a postupně budeme hledat vylepšující cesty. Průběh algoritmu je naznačen na následujících obrázcích (po řádkách). Každý obrázek zachycuje stav po vylepšení toku podél vyznačené vylepšující cesty.

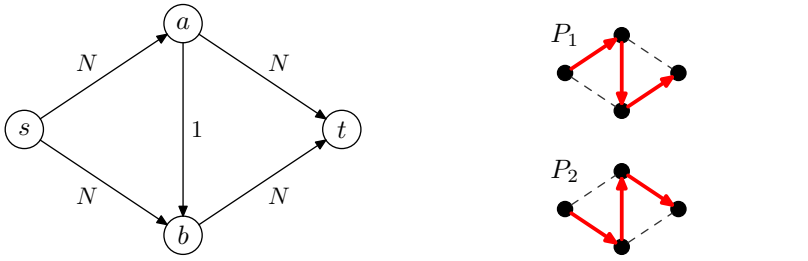






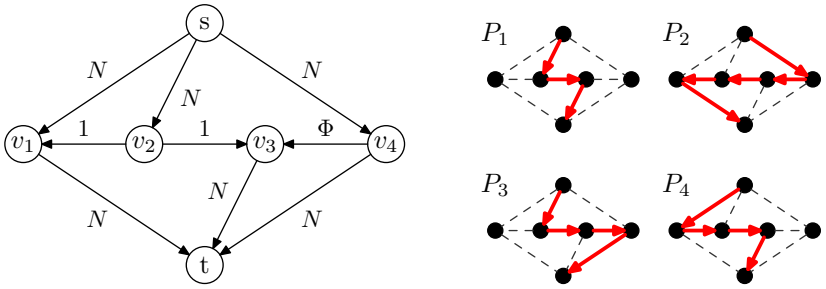
Po čtvrtém vylepšení už vylepšující cesta neexistuje. Proto je nalezený tok velikosti 5 maximální. Pokud bychom chtěli najít minimální řez, tak můžeme postupovat podle návodu z důkazu věty o maximálním toku a minimálním řezu. Vylepšující cesta ze zdroje vede jen do vrcholů $\{s, a, c, d, e\}$ a ty také určují minimální řez velikosti 5.

Příklad: (počet iterací závisí na ohodnocení) Doba běhu algoritmu může značně záviset na velikosti kapacit. Podívejme se na následující síť. Pokud budeme střídavě nacházet vylepšující cestu P_1 a vylepšující cestu P_2, tak budeme muset provést $2N$ vylepšení, než dostaneme maximální tok.



Na příkladu je také vidět, že stačilo vybrat dvě vhodné vylepšující cesty. Jednu vedoucí horem a druhou vedoucí spodem. To nás přivádí k vylepšení, které provedl Edmonds a Karp. Ukázali, že se je výhodné hledat vylepšující cestu s nejmenším počtem hran.

Příklad: (zacyklení, Uri Zwick) Pokud jsou kapacity hran reálná čísla, tak se algoritmus může zacyklit. Uvažme síť na následujícím obrázku. Dvě hrany sítě mají kapacitu jedna, šest hran má kapacitu N, kde N je dostatečně velké číslo, a jedna hrana má kapacitu $\Phi = (\sqrt{5}-1)/2 \approx 0.618$. Číslo Φ je zvoleno tak, aby $1 - \Phi = \Phi^2$. Přenásobením rovnice členem Φ^k dostaneme $\Phi^k - \Phi^{k+1} = \Phi^{k+2}$.



V průběhu Ford-Fulkersonova algoritmu budeme na vodorovných hranách sledovat rezervy po směru hrany a zapisovat je zleva doprava do uspořádané trojice. Připomeňme, že rezerva po směru hrany e je $c(e) - f(e)$.

Začneme s nulovým tokem. Vylepšením toku podél cesty P_1 dostaneme na vodorovných hranách rezervy $(1, 0, \Phi)$. Dále budeme pracovat jednotlivých iteracích. V každé iteraci postupně provedeme čtyři vylepšení podél cest P_2, P_3, P_2, P_4. Předpokládejme, že na začátku iterace jsou rezervy vodorovných hran $(\Phi^{k-1}, 0, \Phi^k)$.

V iteraci postupně zvětšíme tok o Φ^k, Φ^k, Φ^{k+1} a Φ^{k+1}. Rezervy na vodorovných hranách postupně budou

$$\xrightarrow{P_2} (\Phi^{k+1}, \Phi^k, 0) \xrightarrow{P_3} (\Phi^{k+1}, 0, \Phi^k) \xrightarrow{P_2} (0, \Phi^{k+1}, \Phi^{k+2}) \xrightarrow{P_4} (\Phi^{k+1}, 0, \Phi^{k+2})$$

Na konci n-té iterace (to je po $4n+1$ vylepšeních) budou rezervy vodorovných hran Φ^{2n-2}, 0, Φ^{2n-1}. S rostoucím počtem vylepšení konverguje velikost nalezeného toku k hodnotě

$$1 + 2\sum_{i=1}^{\infty} \Phi^i = \frac{2}{1-\Phi} - 1 = 2 + \sqrt{5} < 5.$$

Na druhou stranu je zřejmé, že velikost maximálního toku je $2N + 1$, kde N je libovolně velké číslo.

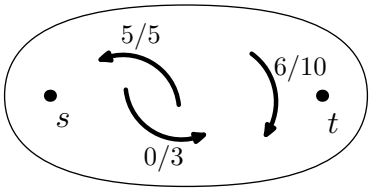
1.2.2 Dinicův/Edmonds-Karpův algoritmus

Pokud budeme ve Ford-Fulkersonově algoritmu volit nejkratší vylepšující cestu (s nejmenším počtem hran), tak se dramaticky zlepší časová složitost celého algoritmu.

Tento nápad uvedli ve své práci už Ford a Fulkerson, ale popsali ho jako heuristiku. Jako první provedl analýzu této heuristiky ruský matematik Dinits (často překládán jako Dinic) v roce 1970. Edmonds a Karp nezávisle publikovali slabší analýzu v roce 1972. Ale protože to byla první anglicky publikovaná analýza, tak se algoritmus často označuje jako Edmonds-Karpův. Dinic navíc přišel s vrstevnatou (čistou) sítí a blokujícím tokem a pomocí něj ukázal rychlejší implementaci algoritmu. Proto budeme algoritmus označovat jako Dinicův.

Ve skutečnosti je stejně těžké najít vylepšující cestu jako najít nejkratší vylepšující cestu. Oboje můžeme řešit průchodem do šířky. Proto je vylepšení algoritmu tak jednoduchou modifikací, že bychom ji ve Ford-Fulkersonově algoritmu použili, aniž bychom o tom věděli.

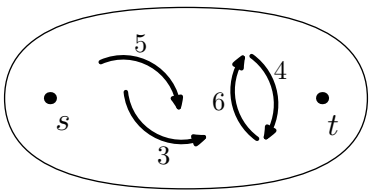
Nejprve si ukážeme, jak jednoduše hledat nejkratší vylepšující cestu.



Původní síť G

1) Máme síť s grafem G a tokem f (*původní síť*). Pro zjednodušení výkladu předpokládejme, že v G ke každé hraně uv existuje opačná hrana vu. Pokud ne, tak do G přidáme hranu vu s nulovou kapacitou. Toto rozšíření grafu G nijak nezmění aktuální, ani maximální tok, ale zjednoduší se zavedení a práci s rezervou hrany.

2) Chceme vytvořit pomocnou síť, která nám zjednoduší hledání vylepšující cesty. Nechceme se dívat na hrany v protisměru, ani nechceme, aby v síti existovaly násobné orientované hrany. Tuto síť vytvoříme na základě původní sítě a toku f. Nazveme ji *síť rezerv* G_f.

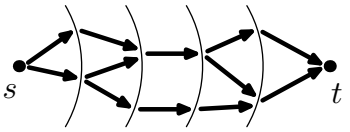


Síť rezerv G_f

Rezerva $r(uv)$ říká, jak velký tok protlačíme z u do v, a odpovídá součtu rezervy hrany uv po směru a rezervy hrany vu v protisměru. Spočítá se jako

$$r(uv) = (c(uv) - f(uv)) + f(vu).$$

Do sítě rezerv dáme jen ty hrany (rozšířené) původní sítě, které mají nenulovou rezervu, a ohodnotíme je rezervou. Každá orientovaná cesta v síti rezerv odpovídá vylepšující cestě v původní síti.



Čistá síť G_f^*

3) Na základě sítě rezerv vytvoříme *čistou síť* G_f^*. Do čisté sítě dáme jen ty hrany sítě rezerv, které leží na nejkratší cestě ze zdroje do spotřebiče. Můžeme ji zkonstruovat pomocí průchodu do šířky, který rozdělí vrcholy do vrstev podle vzdálenosti od zdroje. Proto se této síti někdy říká *vrstevnatá síť.*

Hrana e původní sítě je *nasycená* vzhledem k toku f, pokud $r(e) = 0$ (hranou $e = uv$ i opačnou hranou vu v protisměru protéká největší možný tok směrem z u do v). Orientovaná cesta je *nasycená*, pokud obsahuje nasycenou hranu. Nasycená cesta je tedy opakem vylepšující cesty.

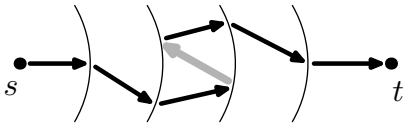
Analýza Dinicova/Edmonds-Karpova algoritmu

Délkou orientované cesty myslíme počet hran na cestě. Vzdálenost z vrcholu x do vrcholu y je délka nejkratší orientované cesty z x do y v síti rezerv G_f. Označíme ji $d_f(x, y)$. Pro cesty vedoucí ze zdroje s píšeme zkráceně $d_f(y)$ místo $d_f(s, y)$.

Klíčové lemma říká, že po vylepšení toku podél nejkratší vylepšující (s, t)-cesty neklesne v síti rezerv délka nejkratší cesty ze zdroje do spotřebiče.

Vylepšením toku podél vylepšující cesty se některé hrany nasytí. Jejich rezerva klesne na nulu a proto zmizí ze sítě rezerv. Tím délka nejkratší cesty v G_f určitě neklesne.

Na druhou stranu se v síti rezerv mohou objevit nové hrany. Jsou to hrany, které měly nulovou rezervu, ale při vylepšení toku jsme po opačné hraně poslali nenulový tok. Každá nová hrana vede z i-té vrstvy čisté sítě do $(i-1)$-ní (pro nějaké i). Každá (s, t)-cesta používající alespoň jednu novou hranu, musí alespoň jednou skočit o vrstvu zpět, ale nikdy nemůže skočit více než o jednu vrstvu dopředu. Proto je nová cesta alespoň o 2 delší, než byla délka cesty, podle které jsme vylepšovali tok.



Lemma zformulujeme malinko obecněji a ukážeme, že po vylepšení neklesne žádná vzdálenost ze zdroje s do libovolného vrcholu v.

Lemma 3 *Nechť f je tok a f' je tok, který vznikne vylepšením f podél nejkratší vylepšující cesty P. Potom pro každý vrchol $v \in V$ platí $d_{f'}(v) \geq d_f(v)$.*

Důkaz: Označme vrcholy na nejkratší vylepšující cestě P v síti G_f jako $s = v_0, v_1, \ldots, v_k = t$. Předpokládejme pro spor, že existuje vrchol v takový, že $d_{f'}(v) < d_f(v)$. Ze všech takových vrcholů si vybereme ten s nejmenším $d_{f'}(v)$. Určitě platí $v \neq s$. Nechť w je předposlední vrchol na nejkratší cestě do v v síti $G_{f'}$, potom $d_{f'}(v) = d_{f'}(w) + 1$. Z volby vrcholu v platí $d_f(w) \leq d_{f'}(w)$.

Hrana wv se musela v grafu objevit až po vylepšení, jinak bychom průchodem po hraně wv v síti G_f dostali $d_f(v) \leq d_f(w) + 1 \leq d_{f'}(w) + 1 \leq d_{f'}(v)$. Proto je wv opačnou hranou k hraně $v_{i-1}v_i$ na cestě P, pro nějaké i. Potom je $d_f(w) = i$ a $d_f(v) = i - 1$. Na druhou stranu je $d_{f'}(v) \geq d_{f'}(w) + 1 \geq i + 1$. To je spor. ∎

Následující lemma říká, že pokud po vylepšení toku podél nejkratší vylepšující (s, t)-cesty nevzroste délka nejkratší (s, t)-cesty, bude nová čistá síť podgrafem původní čisté sítě. Proto stačí aktualizovat původní čistou síť a nemusíme ji po každém vylepšení počítat znova.

Lemma 4 *Nechť f je tok a f' je tok, který vznikne vylepšením f podél nejkratší vylepšující cesty. Pokud $d_f(t) = d_{f'}(t)$, tak $G^*_{f'} \subseteq G^*_f$.*

Důkaz: Čistá síť obsahuje právě hrany ležící na nejkratší cestě ze zdroje do spotřebiče. Nechť $k := d_f(t)$.

Během vylepšování toku podél cesty se mohou objevit nové hrany. Z předchozího důkazu ale vyplývá, že každá cesta ze zdroje do spotřebiče používající novou hranu je alespoň o dva delší než k. Proto se žádná nová hrana nemůže objevit v čisté síti a tedy čistá síť $G^*_{f'}$ je podgrafem předchozí čisté sítě G^*_f. ∎

Lemma 5 *Dinicův/Edmonds-Karpův algoritmus provede vylepšení podél nejvýše mn vylepšujících cest.*

Důkaz: Délka nejkratší vylepšující cesty v průběhu algoritmu neklesá. Proto můžeme běh algoritmu rozdělit do fází podle délky nejkratší vylepšující cesty. Fází je nejvýše tolik, kolik je různých délek cest a to je nejvýše n. Podle lemma 4 do čisté sítě během fáze nepřibudou žádné hrany. V každé fázi provedeme nejvýše m vylepšení, protože se při každém vylepšení nasytí aspoň jedna hrana a zmizí z čisté sítě. ∎

Dinicův algoritmus

Na čistou síť G^*_f s rezervami se můžeme dívat jako na obyčejnou síť s kapacitami (kapacitou každé hrany je velikost rezervy) a můžeme v ní hledat tok. Graf čisté sítě je acyklický orientovaný graf. Tok ϕ v acyklické orientované síti je *blokující tok*, pokud je každá orientovaná (s,t)-cesta v G^*_f nasycená. Důležité je slovo orientovaná. Cesta obsahující hranu v protisměru není přípustná. Blokující tok nemusí být maximální tok, protože může existovat vylepšující cesta používající hrany v protisměru.

Blokující tok můžeme najít pomocí vylepšujících cest, které nevyužívají rezervy v protisměru. Blokující tok ϕ v čisté síti G^*_f je roven toku, o který zvětšíme f během jedné fáze Dinicova algoritmu, tj. při vylepšování toku f podél vylepšujících cest stejné délky.

```
1: Dinic:
2:     zvol počáteční tok, například f := 0
3:     repeat
4:         spočítej síť rezerv
5:         spočítej čistou síť
6:         nalezni blokující tok v čisté síti a přičti ho k f
7:     until spočítaná čistá síť obsahovala hrany
8:     return f
```

Když nově spočítaná čistá síť neobsahuje hrany, tak můžeme skončit, protože neexistuje cesta ze zdroje do spotřebiče v G_f, tedy ani žádná vylepšující cesta v G. V ten moment máme maximální tok.

Repeat-cyklus proběhne nejvýše n-krát, protože v každé iteraci se zvětší délka nejkratší vylepšující cesty. Provedení kroků 4 a 5 bude trvat čas $\mathcal{O}(n+m)$, protože oba kroky provedeme pomocí průchodu grafu. Jak se provede krok 6 si ukážeme za chvilku. Ukážeme, že krok 6 trvá čas $\mathcal{O}(nm)$. Dohromady dostaneme časovou složitost Dinicova algoritmu $\mathcal{O}(n^2m)$.

```
1: nalezení blokujícího toku v čisté síti:
```

```
2:  while čistá síť obsahuje hrany do
3:      najdi v čisté síti cestu ze zdroje do spotřebiče
4:      spočítej hodnotu nejmenší rezervy na cestě
5:      vylepši tok f podél cesty a uprav čistou síť
6:      dočisti čistou síť
```

Krok 3 provedeme hladově například průchodem do hloubky. Při návratu v průchodu do hloubky můžeme rovnou počítat krok 4. Proto budou oba kroky trvat čas $\mathcal{O}(n)$. Stejně tak vylepšení toku podél nalezené cesty.

Jak budeme upravovat a dočisťovat čistou síť? Pro každý vrchol si budeme pamatovat jeho vstupní a výstupní stupeň. Vylepšením toku klesne rezerva některých hran na nulu. Takové hrany musíme z čisté sítě vymazat. Musíme si ale dát pozor, aby nám vymazáním některých hran nevznikly slepé uličky. To jsou cesty vedoucí do vrcholů, ze kterých už nejde pokračovat dál. Proto při vymazávaní každé hrany vložíme její konce do fronty. Při dočišťování čisté sítě postupně probíráme vrcholy ve frontě a pokud mají vstupní nebo výstupní stupeň nula, tak vymažeme všechny hrany z nich vedoucí. Druhé konce mazaných hran vkládáme opět do fronty a při tom aktualizujeme vstupní a výstupní stupně těchto vrcholů. Po zpracování fronty dostaneme korektní čistou síť, ve které můžeme znova začít hledat vylepšující cestu.

Čas za zpracování fronty budeme účtovat jednotlivým hranám. Každý vrchol, který byl vložen do fronty, odpovídá jedné smazané hraně. Hrana mohla do fronty vložit nejvýše své dva koncové vrcholy. Z toho vyplývá, že časová složitost všech provedení kroků 6 je $\mathcal{O}(m)$. While-cyklus proběhne nejvýše m-krát, protože pokaždé vymažeme alespoň jednu hranu čisté sítě. Celková časová složitost nalezení blokujícího toku v čisté síti je $\mathcal{O}(mn)$.

Poznámky k implementaci

Ve skutečnosti nepotřebujeme rozlišovat mezi sítí rezerv a čistou sítí. V iteraci nám stačí jen jedna síť. Nejprve spočítáme síť rezerv. V té provedeme průchod do šířky, během kterého umazáváme hrany neležící na nejkratší cestě ze zdroje do spotřebiče. Nechť S je množina vrcholů ležících na nějaké nejkratší cestě ze zdroje do spotřebiče. Výpočet čisté sítě provedeme následovně. Na začátku je $S = \{t\}$. Síť rezerv procházíme do šířky a při každém návratu po hraně uv hranu smažeme, pokud $v \notin S$, jinak přidáme u do S a hranu necháme v čisté síti. Tím dostaneme čistou síť.

Také není potřeba provádět dokonalé dočišťování čisté sítě. Do vrcholů, do kterých nevede žádná cesta, se nemáme jak dostat. Proto takové vrcholy a cesty z nich vedoucí můžeme v síti nechat.

Odstraňování vrcholů na slepých uličkách, ze kterých nevede cesta dál, můžeme provést podobným trikem, jako při hledání čisté sítě. Čistou sít nebudeme dočišťovat v kroku 6, ale až v kroku 3 při dalším průběhu cyklu. V kroku 3 hledáme cestu ze zdroje do spotřebiče v čisté síti. Hledání provedeme pomocí průchodu do hloubky. Pokud bychom při průchodu do hloubky přešli po hraně uv takové, že z v nelze pokračovat dál, tak při návratu hranu uv smažeme. Mazání naúčtujeme mazaným hranám.

Ve speciálních sítích má Dinicův algoritmus ještě lepší časovou složitost. O tom se ale dozvíte více ve cvičeních. Zájemce také můžeme odkázat na Schrijvera [11] nebo Mareše [9].

1.2.3 Metoda tří Indů

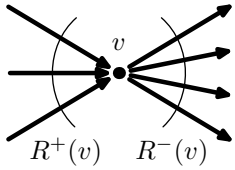
Indové Malhotra, Kumar a Maheshawari v roce 1978 vymysleli efektivnější algoritmus, jak nalézt blokující tok v čisté síti. Jejich metoda běží v čase $\mathcal{O}(n^2)$, což

zlepšuje čas Dinicova algoritmu na $\mathcal{O}(n^3)$.

Pro každý vrchol si spočítáme, jak velký tok může protékat skrz vrchol. Někdy místo „průtok skrz vrchol“ říkáme, jak velký tok jde protlačit skrz vrchol. Největší možný průtok přes vrchol v nazveme rezervou vrcholu v a označíme ho $R(v) := \min\{R^+(v), R^-(v)\}$, kde

$$R^+(v) = \sum_{xv \in E} r(xv),$$

$$R^-(v) = \sum_{vx \in E} r(vx).$$

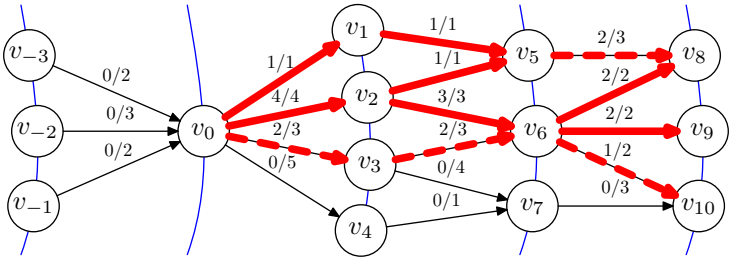


1: nalezení blokujícího toku v čisté síti (podle tří Indů):
2: spočítej $R(v)$ pro každý $v \in V$
3: **while** $V \neq \emptyset$ **do**
4: $v_0 :=$ vrchol $v \in V$ s minimálním $R(v)$
5: **if** $R(v_0) = 0$ **then**
6: $V := V \setminus \{v_0\}$
7: uprav $R(v)$ pro sousedy v_0
8: **else**
9: najdi tok velikosti $R(v_0)$ procházející vrcholem v_0
pomocí protlačení doleva a doprava
a uprav hodnoty $R(v)$

Kroky 5–7 odpovídají pročišťování čisté sítě. Z grafu vyloučíme vrchol v_0 i s hranami, které vedou z v_0 nebo do v_0. Kromě odstranění vrcholů zpracovaných v předchozím průběhu cyklu se takto odstraňují i slepé uličky (rozmyslete si, jak mohou slepé uličky vzniknout a jak je algoritmus odstraní).

Krok 2 bude trvat $\mathcal{O}(m)$. Cyklus proběhne n-krát, protože v každé iteraci vyhodíme z množiny vrcholů jeden vrchol. Ostatní kroky, kromě kroku 9 jsou proveditelné v čase $\mathcal{O}(n)$. Časovou složitost kroku 9 budeme počítat zvlášť a budeme ji účtovat hranám a vrcholům sítě.

Jak probíhá krok 9? Ukážeme si jen protlačování toku doprava. Protlačení toku doleva proběhne symetricky. Protlačování provádíme po vrstvách směrem od v_0.



Na začátku dáme do fronty jen v_0. Vrcholy z fronty postupně zpracováváme následujícím způsobem. Představme si, že už jsme ve vrcholu v, do kterého jsme dotlačili přebytek toku o velikosti K. Postupně probíráme hrany, které vedou z vrcholu v doprava, a snažíme se po nich poslat co největší tok. Pokud bude přebytek toku ve v větší, než rezerva probírané hrany, tak hranu nasytíme a postoupíme k další hraně. Z vrcholu vždy vede další hrana, po které můžeme tok poslat, protože $K \leq R(v_0) \leq R^-(v)$. Druhé konce hran, po kterých jsme poslali nějaký tok, vložíme do fronty.

Práci s hranami, které jsme nasytili, naúčtujeme hranám (nasycené hrany zmizí z čisté sítě). Práci s poslední hranou, po které jsme poslali nějaký tok, ale nemuseli jsme ji nasytit, naúčtujeme vrcholu v. Celkem jsme během algoritmu naúčtovali

každé hraně nejvýše jednu jednotku práce a každému vrcholu nejvýše n jednotek práce. Proto je celková časová složitost kroku 9 rovna $\mathcal{O}(n^2 + m)$.

Časová složitost celého algoritmu na nalezení blokujícího toku podle metody tří Indů je $\mathcal{O}(n^2)$. To dává časovou složitost nalezení maximálního toku $\mathcal{O}(n^3)$.

Poznámka k implementaci

Můžeme se vyhnout použití fronty při protlačování toku. Během kroku 2 si v čase $\mathcal{O}(m)$ spočítáme topologické uspořádání vrcholů čisté sítě (topologické uspořádání hledáme pro čistou síť pouze jednou). Během protlačování toku doprava postupně v topologickém pořadí probíráme vrcholy, které jsou topologicky větší než v_0 a pokud mají kladný přebytek toku, tak přebytek protlačíme do sousedních vrcholů. Podobně při protlačování toku doleva.

1.3 Goldbergův Push-Relabel algoritmus

V algoritmech vylepšující cesty se tok podél jedné hrany postupně nasčítává z toků podél vylepšujících cest. Těchto cest může být poměrně mnoho a nalezení zlepšující cesty může trvat až $\mathcal{O}(n)$. Proto se naskýtá myšlenka, jestli nemůžeme tok podél hrany poslat naráz.

Ukážeme si algoritmus, který je založen na daleko jednodušší myšlence než jsou vylepšující cesty. Algoritmus používá dvě základní operace: protlačení toku po hraně (push)[6] a zvýšení výšky vrcholu (relabel). Proto se algoritmus nazývá push-relabel. Algoritmus vymyslel Goldberg v roce 1985. Varianta, kterou si ukážeme je podle Goldberga a Tarjana z roku 1988.

Připomeňme, že G je orientovaný graf, jehož hrany jsou ohodnoceny kapacitami $c : E \to \mathbb{R}_+$. Graf G rozšíříme tak, aby ke každé hraně uv existovala opačná hrana vu. Přidávané hrany budou mít kapacitu nula, takže nijak neovlivní tok v síti (ve skutečnosti žádné hrany přidávat nemusíme, používáme je jen pro zjednodušení definice rezervy). Rozšířenému grafu budeme říkat původní síť.

Budeme pracovat s pomocným grafem G_f (síť rezerv) podobně jako v algoritmech pracujících s vylepšující cestou. Pro dvojici G a f dáme do grafu G_f každou hranu původního grafu s nenulovou rezervou. Připomeňme, že rezerva hrany uv je $r(uv) = (c(uv) - f(uv)) + f(vu)$. Rezerva $r(uv)$ znamená, že můžeme skrz hranu uv, případně hranu vu, *protlačit* $r(uv)$ jednotek toku z vrcholu u do vrcholu v. Aby v původním grafu po dvojici hran uv, vu neproudil tok tam i zpět, tak nejprve protlačíme co největší část toku v protisměru po vu a pak teprve zbytek toku po uv. Změnou toku se změní i pomocný graf G_f.

Dále připomeňme, že $f(v)$ značí *bilanci vrcholu* v, nebo také *přebytek* toku ve vrcholu v.

Myšlenka protlačování

Nejprve zavedeme jeden klíčový pojem. Funkce $f : E \to \mathbb{R}_+$ je *pratok*[7], pokud splňuje

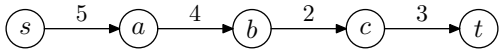
$$\begin{array}{lll} i) & 0 \le f(e) \le c(e) & \text{pro každou hranu } e \in E \\ ii) & f(v) \ge 0 & \text{pro každý vrchol } v \in V \setminus \{s,t\} \end{array}$$

Řekneme, že vrchol $v \in V \setminus \{s,t\}$ je *aktivní,* pokud $f(v) > 0$. Tedy pokud do vrcholu přitéká více, než z něj odtéká. Vrcholy s, t nejsou nikdy aktivní.

Jak se liší pratok od toku? V definici toku platí druhá podmínka s rovností ($f(v) = 0$ pro každý vrchol $v \in V \setminus \{s,t\}$).

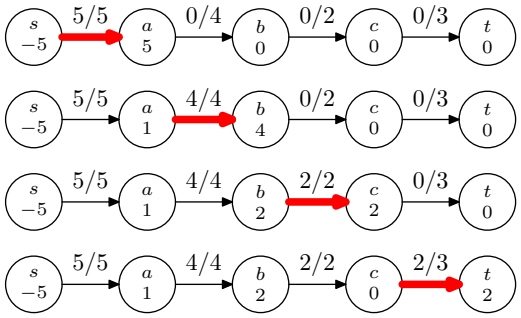
Podívejme se na základní myšlenku push-relabel algoritmu. Nejprve protlačíme ze zdroje co největší tok do sousedních vrcholů. Dále budeme probírat aktivní vrcholy a snažit se protlačit přebytek toku v nich do sousedních vrcholů směrem ke spotřebiči. Při protlačování toku nesmíme překročit kapacitu hran. Protlačování bude probíhat pouze po hranách s nenulovou rezervou. Postupně budeme chtít protlačit všechny přebytky toku až do spotřebiče. Když to nepůjde, tak je protlačíme zpátky do zdroje.

Podívejme se na příklad sítě na obrázku, ve které najdeme maximální tok pomocí protlačování toku po hranách.

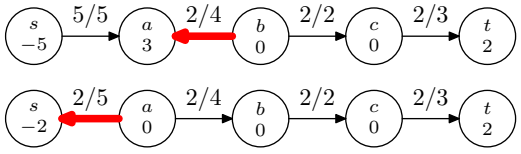


[6] Protlačování toku po hraně je podobné protlačování toku v metodě tří Indů.
[7] z anglického preflow

Průběh protlačování toku si můžeme představit jako vlnu, která se šíří ze zdroje do spotřebiče. Tam se odrazí a valí se zpátky do zdroje. Jednotlivé kroky protlačování jsou na následujících obrázcích. U každé hrany na obrázku je hodnota „$f(e)/c(e)$“. Ve vrcholech je uvedeno jméno vrcholu a aktuální přebytek toku. V první fázi postupně protlačujeme co nejvíce toku ze zdroje směrem ke spotřebiči. Ve zdroji s je nekonečně mnoho vody a tak po hraně sa protlačíme 5 jednotek toku do a. V dalším kroku protlačíme po hraně ab co největší část přebytku $f(a)$. Dále postupujeme podobně a to tak dlouho, dokud nedorazíme do spotřebiče.



Do spotřebiče jsme dotlačili největší možný tok, ale vrcholy a, b jsou stále aktivní. V grafu G_f nevede orientovaná cesta z aktivních vrcholů do spotřebiče t a proto nastane druhá fáze, ve které dotlačíme přebytek toku z aktivních vrcholů zpátky do zdroje.



Skončili jsme s maximálním tokem.

Na cestě je hledání toku jednoduché, ale v jakém pořadí provádět jednotlivá protlačení v obecném grafu?

Musíme si dát pozor, abychom se nezacyklili. Například by se mohlo stát, že budeme neustále protlačovat tok po jedné hraně tam a zpátky, tam a zpátky, ...

Rozhodování, podél kterých hran budeme tok protlačovat, provedeme na základě odhadu vzdáleností v G_f. Přebytky toku budeme protlačovat po nejkratších cestách do spotřebiče. Za chvíli si vysvětlíme, co je to výška každého vrcholu. Dolním odhadem vzdálenosti dvou vrcholů potom bude jejich výškový rozdíl. Samotný algoritmus nebude přemýšlet nad nejkratšími cestami do spotřebiče, ale bude tok posílat po libovolné hraně vedoucí z kopce dolů.

Platné označkování a výšky vrcholů

Vektor $d \in (\mathbb{N}_0 \cup \{\infty\})^n$ nazveme *platné označkování*[8] vrcholů vzhledem k toku f, pokud

$$\begin{aligned} &i) \quad d(s) = n, \quad d(t) = 0, \\ &ii) \quad d(v) \leq d(w) + 1 \qquad \text{pro každou hranu } vw \in E(G_f). \end{aligned}$$

Pod hodnotou $d(v)$ si budeme představovat *výšku,* ve které se vrchol v nachází. Protlačování toku budeme provádět podél hran vedoucích z kopce dolů. To je tak, jak voda přirozeně teče. Platné označkování říká, že zdroj bude vždy ve výšce n,

[8] z anglického valid labeling

spotřebič ve výšce 0. Neklade žádná omezení na stoupání, ale říká, že žádná hrana G_f nevede příliš strmě dolů.[9] Hrana může klesat nejvýše o jedna.

Z existence platného označkování vrcholů vyplývá důležitá vlastnost pratoku, která říká, že pratok „nasycuje" jistý řez. Řez $\delta(R)$ je *nasycený,* pokud pro každou hranu $e \in \delta(R)$ je $f(e) = c(e)$ a pro každou hranu $\delta(\overline{R})$ je $f(e) = 0$. Připomeňme, že tok je vždy menší roven velikosti řezu. Pokud pro tok f najdeme řez $\delta(R)$ nasycený tokem f, tak víme, že je tok maximální (platí $c(\delta(R)) = |f|$).

Lemma 6 *Nechť f je pratok a d je platné označkování pro f. Potom existuje nasycený (s,t)-řez $\delta(R)$.*

Důkaz: Protože má graf G_f jen n vrcholů, tak existuje hodnota k, $0 < k < n$, taková, že $d(v) \neq k$ pro všechny vrcholy $v \in V$. Položme $R = \{v \in V : d(v) > k\}$. Potom $s \in R$ a $t \notin R$, protože $d(s) = n$ a $d(t) = 0$. Z bodu *ii)* definice platného označkování plyne, že žádná hrana G_f nevede z R ven, protože nemůže klesnout o více než jedna. ∎

Důsledek 4 *Pokud existuje platné označkování pro tok f, tak je tok f maximální.*

Důsledek nám dává podmínku pro zastavení algoritmu. Push-relabel algoritmus si neustále udržuje platný pratok a platné označkování (tedy i nasycený řez). Skončí v momentě, kdy se z pratoku stane tok. V jistém smyslu je duální k algoritmům vylepšující cesty, protože ty si udržují platný tok a skončí, až se některý řez nasytí.

Připomeňme, že $d_f(v,w)$ je orientovaná vzdálenost v grafu G_f, tj. počet hran na nejkratší orientované cestě z v do w.

Ukážeme si, že rozdíl výšek dvou vrcholů je dolním odhadem jejich vzdálenosti v G_f.

Lemma 7 *Nechť f je pratok a d platné označkování. Potom pro každé dva vrcholy $v, w \in V$ platí $d_f(v,w) \geq d(v) - d(w)$.*

Důkaz: Pokud je $d_f(v,w) = \infty$, tak lemma platí. Předpokládejme tedy, že je $d_f(v,w)$ konečné. Uvažme nejkratší orientovanou (v,w)-cestu v G_f. Pro každou hranu pq orientované cesty z definice platného označkování platí $d(p) - d(q) \leq 1$. Sečtením těchto nerovností podél hran orientované cesty dostaneme výsledek.[10] ∎

Ve speciálním případě lemma říká, že

- $d(v)$ je dolním odhadem na $d_f(v,t)$ a že
- $d(v) - n$ je dolním odhadem na $d_f(v,s)$.

Poznamenejme, že $d(v) \geq n$ znamená, že $d_f(v,t) = \infty$. Tedy že v G_f nevede orientovaná cesta z v do spotřebiče t a proto by se měl v tomto případě posílat přebytek toku ve v zpátky do zdroje. Bez ohledu na velikost $d(v)$ budeme protlačovat tok z kopce dolů. Tedy z vrcholu v do vrcholů w s $d(w) < d(v)$, protože se tím podle odhadů dostane tok z v blíže k místu určení.

Poznámka: (o výpočtu platného označkování) Důkaz lemmatu nám naznačuje, jak si spočítat platné označkování, pokud bychom ho neznali. Nejprve se podívejme na speciální případ, kdy ze všech vrcholů kromě s existuje v síti rezerv G_f orientovaná cesta do t. Za výšky $d(v)$ zvolíme délku nejkratší cesty z v do t, jinými slovy $d(v) := d_f(v,t)$. Potom označkování vrcholů $d(v)$ splňuje vlastnosti platného

[9] Nasycené hrany zmizí ze sítě rezerv G_f. Proto se na ně nevstahuje omezení klesání.

[10] Z důkazu je vidět, odkud se vzala druhá podmínka v definici platného označkování. Potřebujeme, aby tohle lemma platilo.

označkování až na podmínku $d(s) = n$. Hodnotu $d(s)$ si můžeme zvolit jak chceme, protože v síti G_f nevede z s žádná hrana do ostatních vrcholů. Jinak by v G_f existovala i cesta z s do t. (Všechny hrany vedoucí z s jsou nasycené a tudíž nejsou v síti G_f).

V obecném případě zvolíme $d(v) := \min\{d_f(v,t),\ n + d_f(v,s)\}$. Důkaz, že takto dostaneme platné označkování, necháme jako cvičení. Poznamenejme jenom, že množina vrcholů, ze kterých v G_f vede orientovaná cesta do t, určuje nasycený řez.

Výpočtu platného označkování využijeme později v heuristice na straně 23.

Push-Relabel algoritmus

Inicializace: Začneme s počátečním pratokem f takovým, že $f(e) = c(e)$ pro hrany $e \in E$ vedoucí ze zdroje s a $f(e) = 0$ pro ostatní hrany. Položme $d(s) = n$ a $d(v) = 0$ pro všechny ostatní vrcholy v. Označkování d je platným označkováním pro pratok f, protože všechny hrany G_f mají oba konce ve výšce nula.

Hlavním úkolem ve zbytku algoritmu je likvidovat aktivní vrcholy, protože až v grafu nebude existovat aktivní vrchol, tak se pratok stane tokem. Při zpracování vrcholů nám pomohou následující operace.

Operace protlač: Operaci protlačení toku po hraně $vw \in E(G_f)$ nazveme *protlač*(vw) (anglicky se nazývá *push*). Jak už jsme řekli, přebytek toku budeme protlačovat pouze po hranách vedoucích z kopce dolů. Tedy při protlačení toku po hraně $vw \in E(G_f)$ je $d(w) < d(v)$. Protože hrany sítě rezerv nemohou klesat nejvíce o jedna, musí být $d(v) = d(w) + 1$. Abychom měli co protlačovat, tak musí být ve v kladný přebytek toku. To znamená, že v musí být aktivní vrchol. Protlačování proto může probíhat pouze po hranách, které vedou z aktivních vrcholů a klesají právě o jedna. Takové hrany nazveme *přípustné.*

Hrana uv se při protlačení buď nasytí a zmizí z G_f, nebo se nenasytí a zůstane v G_f. Do sítě rezerv přibude zpětná hrana wv, která vede do kopce. Jiné změny v G_f nejsou a proto po protlačení toky po vw zůstane označkování d platné.

Operace zvýšení: Předpokládejme, že v je aktivní, ale z v už v G_f nevede žádná přípustná hrana. Potom můžeme zvýšit $d(v)$ na $\min\{d(w) + 1 \mid vw \in E(G_f)\}$, aniž bychom porušili platnost označkování. Této operaci budeme říkat *zvýšení* vrcholu v (anglicky se označuje *relabel* nebo *lift*). Jinými slovy, aktivní vrchol v zvedáme o jedničku tak dlouho, dokud některá hrana z vrcholu vycházející nepovede z kopce.

Je spousta výsledků o tom, v jakém pořadí se mají operace provádět. My si předvedeme obecnější verzi algoritmu. Jakmile vybereme aktivní vrchol v, tak budeme provádět operaci protlač po přípustných hranách G_f tak dlouho, dokud se vrchol v nestane neaktivním a nebo dokud ho nezvýšíme. Tuto posloupnost operací označíme jako zpracování vrcholu.

```
Zpracuj(v):
    while v je aktivní a existuje přípustná hrana vw ∈ E(G_f) do
        Protlač co největší část přebytku ve v po hraně vw
    if v je aktivní then
        Zvyš vrchol v
```

Samotný algoritmus potom můžeme vyjádřit následovně.

```
Push-Relabel:
    Inicializuj f a d
    while f není tok do
        Vyber aktivní vrchol v
        Zpracuj v
```

Je několik různých pravidel pro výběr aktivního vrcholu.

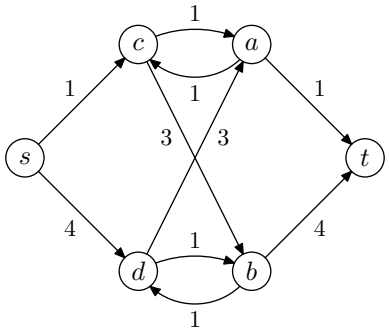
- Vybereme vrchol v s maximálním označením $d(v)$. Algoritmus s tímto pravidlem se označuje jako *maximum distance push-relabel.*[11] Toto pravidlo se používá nejčastěji, protože garantuje nejlepší časovou složitost algoritmu.

- Aktivní vrcholy vkládáme do fronty. Aktivní vrcholy zpracováváme v pořadí, jak jsou ve frontě. (Pokud vrchol zůstal po zpracování aktivní, tak se ocitne na konci fronty). Algoritmus s tímto pravidlem se označuje jako *FIFO push-relabel.*

Algoritmus si v průběhu udržuje platný pratok a také platné označkování. Při popisu operací jsme ověřili, že se provedením operace platnost označkování nezmění. Proto z důsledku 4 dostáváme, že pokud nebude existovat aktivní vrchol, tak se algoritmus zastaví a skončí s maximálním tokem.

Příklad

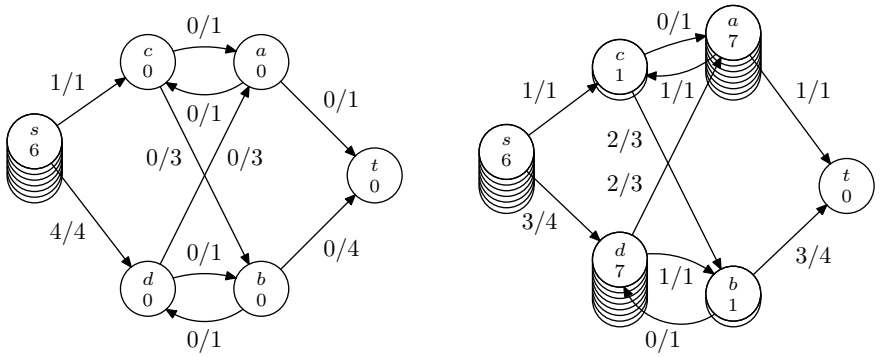
Na následujícím obrázku je graf, ve kterém chceme najít maximální tok pomocí algoritmu push-relabel. Použijeme pravidlo, které si vždy vybere aktivní vrchol s největším $d(v)$. Pokud mají dva vrcholy stejnou hodnotu $d(v)$, tak vybereme ten abecedně menší. V každém vrcholu v probíráme přípustné hrany vw v abecedním pořadí podle w.



Nejprve provedeme inicializaci a dostaneme ohodnocený graf na následujícím obrázku vlevo. Pozor, hodnota u vrcholu v není rezerva, ale výška $d(v)$. Po inicializaci jsou vrcholy c, d aktivní. Vybereme si c a zvýšíme $d(c)$ na 1. V dalším kroku protlačíme po hraně ca tok velikosti 1. Po tomto kroku už jsou všechny kroky algoritmu jasně určeny. Doporučujeme čtenáři, aby si odkrokoval zbytek algoritmu. Aktivní vrcholy zvolené algoritmem jsou c, a, d, d, a, a, d, a, d, a, d, a, d, c, b a postupně algoritmus protlačuje tok po hranách ca, at, db, da, ac, ad, da, ad, da, ad, ds, cb, bt (aktivní vrchol většinou zvedáme o 1, párkrát o 2 a v předposledním případě vůbec). Pro lepší pochopení si v každém kroku načtněte graf G_f, ať vidíte, které hrany jsou nasycené a vypadly. Algoritmus skončí s tokem a označkováním na následujícím obrázku vpravo.[12]

[11] Maximum distance proto, že hodnota $d(v)$ se označuje jako distance label.

[12] Všimněme si zdlouhavého protlačování toku mezi vrcholy a,d. Nejprve protlačíme přebytek toku z a do d, pak ten samý tok zpátky z d do a a tak několikrát dokola. Vypadá to jako „ping-pong“, který hrajeme tak dlouho, dokud výška jedno z vrcholů nepřekročí 6. V průběhu „ping-pongu“ v G_f neexistuje cesta do spotřebiče t. Proto kdybychom rovnou zvedli výšky vrcholů a, d na 6, tak bychom neporušili platnost označkování a ušetřili si „ping-pong“. Jak této myšlenky využít se dozvíte v heuristice na straně 23.



Výsledné označkování neobsahuje žádný vrchol s $d(v) = 5$ a proto množina $R = \{s, d, a\}$ určuje minimální řez (viz lemma 6).

Analýza Push-Relabel algoritmu

Naším cílem je dokázat následující věty.

Věta 2 *Algoritmus push-relabel provede $\mathcal{O}(n^2)$ zvýšení vrcholů a $\mathcal{O}(mn^2)$ protlačení po hraně.*

Věta 3 *Algoritmus maximum distance push-relabel provede $\mathcal{O}(n^2)$ zvýšení vrcholů a $\mathcal{O}(n^3)$ protlačení po hraně.*

První větu dokážeme řadou lemmat. Lemmata platí pro obecný push-relabel algoritmus. Akorát lemma 12 platí pouze pro maximum distance pust-relabel algoritmus. Jeho přidáním k předchozím lemmatům dokážeme větu 3.

Lemma 8 *Je-li f je pratok a w je aktivní vrchol, potom v G_f existuje orientovaná cesta z w do zdroje s.*

Důkaz: Sporem, nechť z w nevede orientovaná cesta do s. Označme jako R množinu vrcholů, ze kterých v G_f vede orientovaná cesta do zdroje s. Potom v G_f nevede žádná hrana z $\overline{R}$ do R a proto $f(\delta(R)) = 0$. Sečtěme nerovnosti $f(v) \geq 0$ pro všechny $v \in \overline{R}$ a dostaneme $X := \sum_{v \in \overline{R}} f(v) \geq 0$. Na druhou stranu, když sečteme příspěvky přebytků po hranách, tak dostaneme $X = f(\delta(R)) - f(\delta(\overline{R}))$ (každá hrana uv s oběma konci v $\overline{R}$ přispěje do $f(v)$ kladně a do $f(u)$ záporně a proto je její příspěvek do X roven nule). Protože $f(\delta(R)) = 0$, tak z nerovnosti $f(\delta(R)) - f(\delta(\overline{R})) \geq 0$ dostáváme $f(\delta(\overline{R})) = 0$. Součet nerovností platí s rovností a proto i každá nerovnost platí s rovností. To je spor s tím, že pro $w \in \overline{R}$ je $f(w) > 0$. ∎

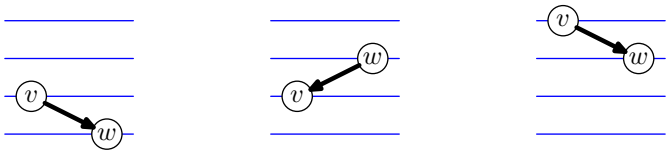
Lemma 9 *V každém kroku algoritmu pro každý vrchol $v \in V$ je $d(v) \leq 2n - 1$. Každý vrchol se zvýší nejvýše $2n - 1$ krát a tedy celkem proběhne nejvýše $\mathcal{O}(n^2)$ zvýšení.*

Důkaz: Každé zvýšení proběhne alespoň o jedna. Zvyšovány jsou pouze aktivní vrcholy a z těch vždy vede cesta do zdroje (lemma 8). Tedy $d_f(v, s) \leq n - 1$. Z lemmatu 7 víme, že $d_f(v, s) \geq d(v) - n$. Kombinací obou nerovností dostaneme $d(v) \leq 2n - 1$. ∎

Operace protlačení rozdělíme na dva typy podle toho, jestli se při ní hrana nasytila a nebo ne. Protlačení po hraně vw je *nasycující*, pokud byl přebytek ve v větší než rezerva hrany vw. Po nasycujícím protlačení zmizí hrana vw z G_f. V opačném případě je protlačení po hraně vw *nenasycující* a v tomto případě přestal být vrchol v aktivní.

Lemma 10 *Počet nasycujících protlačení během algoritmu je nejvýše* $2mn$.

Důkaz: Podívejme se na pevný pár (v, w) takový, že $vw \in E$ nebo $wv \in E$. Po nasycujícím protlačení po hraně vw hrana vw zmizí z G_f. Proto mezi dvěma nasycujícími protlačeními po hraně vw musí proběhnout protlačení po opačné hraně wv, aby se hrana vw opět objevila v G_f.



Protlačování probíhá pouze po přípustných hranách, vedoucích z kopce o jedna dolů. Hodnota $d(v)$ nikdy neklesá a proto musí být mezi dvěma nasycujícími protlačeními po vw alespoň jedna operace zvýšení vrcholu v. Zvýšení $d(v)$ bude alespoň o 2 a proto podle lemmatu 9 může proběhnout nejvýše $n-1$ krát. Celkem po hraně vw může proběhnout nejvýše n nasycujících protlačení.

Podobně po opačné hraně wv může proběhnout také nejvýše n nasycujících protlačení. Celkem tedy pro všechny hrany v původní síti G proběhne nejvýše $2mn$ nasycujících protlačení. ∎

Lemma 11 *Počet nenasycujících protlačení během algoritmu je nejvýše* $\mathcal{O}(mn^2)$.

Důkaz: Nechť A je množina aktivních vrcholů vzhledem k pratoku f a $D = \sum_{v \in A} d(v)$. Na začátku algoritmu je $D = 0$ a nikdy není záporné.

- Každé zvýšení vrcholu zvětšuje D.
- Nasycující protlačení po hraně vw může zvětšit D až o $2n-1$, protože v může zůstat aktivní a vrchol w se může stát aktivním. Podle lemma 9 je $d(w) \leq 2n-1$.
- Nenasycující protlačení po vw zmenší D. Vrchol v přestane být aktivní. Proto se D zmenší buď o $d(v)$ nebo o $d(v) - d(w) = 1$, pokud se zároveň w stane aktivním.

Všechny zvětšení D během algoritmu jsou kvůli zvýšení vrcholů a nebo kvůli nasycujícím protlačením. Podle lemma 9 a lemma 10 během celého algoritmu vzroste hodnota D o nejvýše $(n-2)(2n-1) + 2mn(2n-1) = \mathcal{O}(mn^2)$. Každý nenasycující protlačení sníží tuto hodnotu alespoň o jedna a proto proběhne nejvýše $\mathcal{O}(mn^2)$ nenasycujících protlačení. ∎

Lemma 12 *Počet nenasycujících protlačení během maximum distance push-relabel algoritmu je nejvýše* $\mathcal{O}(n^3)$.

Důkaz: Každé nenasycující protlačení po vw deaktivuje vrchol v. Protože vždy zpracováváme vrchol v s největším $d(v)$, tak je $d(w) \leq d(v)$ pro všechny aktivní vrcholy w. Před tím, než se v stane znovu aktivním, musí proběhnout zvýšení nějakého souseda v (a protlačení toku z tohoto souseda do v).

Z toho dostáváme, že když proběhne n nenasycujících protlačení a žádné zvýšení vrcholu, tak žádný vrchol není aktivní a algoritmus skončí. Proto je počet nenasycujících protlačení nejvýše n krát větší než počet zvýšení a to je nejvýše $\mathcal{O}(n^3)$. ∎

Poznamenejme, že jsou i další pravidla pro výběr aktivních vrcholů a dávají také odhad $\mathcal{O}(n^3)$ na celkový počet protlačení (například FIFO push-relabel). My jsme si vybrali pravidlo maximum distance z toho důvodu, že pro něj Tunçel v roce 1994 dokázal lepší analýzu a ukázal odhad $\mathcal{O}(n^2\sqrt{m})$ na počet protlačení (viz lemma 13).

Tunçelův odhad na počet nenasycujících protlačení*

Lemma 13 *Počet nenasycujících protlačení během maximum distance push-relabel algoritmu je nejvýše* $\mathcal{O}(n^2\sqrt{m})$.

Důkaz: Připomeňme, že $d(v)$ nazýváme výškou vrcholu v. V průběhu algoritmu H označuje maximální výšku aktivního vrcholu.

Výpočet algoritmu rozdělíme do fází mezi změnami hodnoty H. Změna fáze nastane když dojde ke zvýšení vrcholu, který ležel ve výšce H, a nebo když se přebytky všech vrcholů ležících ve výšce H sníží na nulu.

Nyní ukážeme, že celkem proběhne nejvýše $4n^2$ fází. $H \geq 0$, roste pouze při zvýšení vrcholu a to o jedna. Celkový počet zvýšení H je nejvýše počet zvýšení vrcholů a to je nejvýše $2n^2$ (lemma 9). Počet poklesů H je nejvýše tolik, kolik celkem proběhne zvýšení H. Celkový počet změn H (zvýšení nebo poklesů) a tedy i počet fází je nejvýše $4n^2$.

Počet nenasycených protlačení spočítáme pomocí potenciálu. Pevně si zvolíme parametr $K \in \mathbb{N}$. Později ukážeme, že optimální volba je $K := \sqrt{m}$. Zvolme potenciál

$$\Psi := \sum_{f(v)>0} \frac{\vartheta(v)}{K},$$

kde $\vartheta(v) := |\{w \in V \mid d(w) \leq d(v)\}|$ je počet vrcholů ve výškách nejvýše $d(v)$. Fázi nazveme *levnou*, pokud během ní proběhne nejvýše K nenasycených protlačení, a *drahou* jinak.

Levných fází je nejvýše tolik, kolik je všech fází a to je nejvýše $4n^2$. Celkem během levných fází proběhne nejvýše $4Kn^2$ nenasycených protlačení.

Teď odhadneme počet nenasycujících protlačení během drahých fází. Podívejme se, co se děje s potenciálem Ψ při následujících operacích.

- Zvýšení vrcholu. Při zvýšení vrcholu v se $\vartheta(v)$ zvýší nejvýše o n. $\vartheta(w)$ ostatních vrcholů w může jen klesnout (vždy zvyšujeme vrchol v s maximálním $d(v)$). Proto se Ψ zvýší nejvýše o n/K.

- Nasycující protlačení po hraně uv. Protože neměníme výšky vrcholů, tak můžeme Ψ ovlivnit pouze tím, že přibude nebo ubude aktivní vrchol. Můžeme ubrat sčítanec $\vartheta(u)/K$ a přidat $\vartheta(v)/K \leq n/K$. Nasycující protlačení tedy zvýší Ψ nejvýše o n/K. Podle lemma 11 je počet nasycujících protlačení během celého algoritmu nejvýše $2mn$.

- Nenasycující protlačení po hraně uv. Po protlačení bude $f(u) = 0$ a tedy z Ψ ubyde $\vartheta(u)/K$. Dále může přibýt $\vartheta(v)/K$. Celkový úbytek Ψ bude nejvýše $(\vartheta(u) - \vartheta(v))/K$. Protože $H = d(u) = d(v) + 1$ (uv je přípustná hrana), tak $(\vartheta(u) - \vartheta(v))$ odpovídá počtu vrcholů ve výšce H. Počet vrcholů ve výšce H se během fáze nemění. Nemůže klesnout, protože pouze zvyšujeme vrcholy a povýšením některého vrcholu na výšku $H + 1$ ukončíme fázi. V průběhu fáze dojde nejvýše k tolika nenasycujícím protlačení, kolik je aktivních vrcholů ve výšce H. V drahé fázi proběhne alespoň K nenasycujících protlačení. Matematicky vyjádřeno $K \leq$ #nenasycujících protlačení $\leq$ #vrcholů ve výšce $H = \vartheta(u) - \vartheta(v)$. Proto je $(\vartheta(u) - \vartheta(v))/K \geq 1$. Tedy nenasycující protlačení sníží Ψ alespoň o 1.

Celkový součet přírůstků Ψ je nejvýše $(2n^2+2nm)n/K$. Po inicializaci algoritmu bylo Ψ nejvýše n^2/K. Protože Ψ je stále kladné a každé nenasycující protlačení sníží Ψ alespoň o 1, je počet nenasycujících protlačení v drahých fázích nejvýše

$$n^2/K + (2n^2 + 2nm)n/K \leq 5n^2m/K.$$

Při odhadu jsme použili nerovnost $n \leq m$. Dohromady je počet nenasycujících protlačení v levných i drahých fázích nejvýše

$$4n^2K + 5n^2m/K \leq 5n^2(K + m/K) \leq 10n^2\sqrt{m}$$

pro volbu $K = \sqrt{m}$. ∎

Implementace algoritmu Push-Relabel

Zatím jsme ukázali odhady na počet provedení operací zvýšení vrcholu a protlačení toku po hraně. Abychom mohli něco tvrdit o časové složitosti, tak ještě musíme upřesnit, jak proběhne nalezení přípustné hrany a jak poznáme, že je čas zvýšit vrchol. O maximum distance push-relabel algoritmu budeme muset ještě ukázat, jak najít aktivní vrchol v s maximálním $d(v)$.

Pozorování 1 *Nechť $v \in V$ je aktivní a hrana vw není přípustná. Před tím, než se hrana vw stane přípustnou, bude muset proběhnout zvýšení vrcholu v.*

Důkaz: Pokud hrana vw není přípustná, tak buď $d(v) \leq d(w)$ a nebo $r(vw) = 0$. Druhý případ se může změnit pouze protlačením toku po opačné hraně wv, ale potom bude $d(w) = d(v) + 1$. Proto bude v obou případech platit $d(v) \leq d(w)$ a jenom zvýšení vrcholu v to může změnit. ∎

Pro každý vrchol si pamatujeme seznam sousedů $Sousedi(v)$. Zpracování vrcholu v proběhne tak, že postupně projdeme $w \in Sousedi(v)$ a provedeme protlačení po přípustných hranách vw. Průchod seznamu sousedů skončí buď tak, že se v stane neaktivním, nebo tím, že dojdeme na konec seznamu $Sousedi(v)$.

V momentě, kdy dorazíme na konec seznamu, tak už z v nevede žádná přípustná hrana a proto zvýšíme v. Ke zvýšení vrcholu potřebujeme znát minimum z $d(w)$ pro všechny $w \in Sousedi(v)$. Toto minimum si můžeme počítat už během průchodu seznamu sousedů. Zvýšení vrcholu v dokonce proběhne právě tehdy, když se dostaneme na konec seznamu $Sousedi(v)$.

Když bude v znovu vybrán ke zpracování, tak podle pozorování nemohly od posledního zvýšení v přibýt nové přípustné hrany. Proto při zpracování vrcholu v nemusíme procházet seznam $Sousedi(v)$ od začátku, ale můžeme začít tam, kde jsme naposledy skončili (pro každý vrchol musíme pamatovat aktuální pozici v seznamu sousedů).

Protože každý vrchol můžeme zvýšit nejvýše $2n - 1$ krát, tak projdeme seznam sousedů každého vrcholu také nejvýše $2n - 1$ krát. Z toho důvodu je celkový čas strávený hledáním přípustných hran roven $\mathcal{O}(\sum_{v \in V} n \cdot |Sousedi(v)|) = \mathcal{O}(nm)$. Celkový čas strávený nad zvyšováním vrcholů je stejný.

Celkový čas strávený nad operacemi protlačení je $\mathcal{O}(N)$, kde N je počet všech protlačení. Čas nalezení dalšího aktivního vrcholu je konstantní, protože nám nezáleží na pořadí aktivních vrcholů a můžeme si je dávat do fronty. Aktivní vrchol hledáme nejvýše tolikrát, kolik je všech operací protlačení a zvýšení vrcholu. Celkem tedy hledání aktivních vrcholů zabere čas $\mathcal{O}(N + n^2)$ a to je podle věty 2 nejvýše $\mathcal{O}(n^2m)$.

Právě jsme si ukázali, že push-relabel algoritmus může být implementován tak, aby běžel v čase $\mathcal{O}(n^2m)$.

Podívejme se na případ maximum distance push-relabel algoritmu. Není jasné, jak implementovat výběr aktivního vrcholu s maximálním $d(v)$ tak, aby celkem

běžel v čase $\mathcal{O}(N)$. To v průměru odpovídá konstantnímu času na jednu operaci protlačení. Jednoduché řešení projde všechny vrcholy, ale to trvá čas $\mathcal{O}(n)$ na jedno nalezení aktivního vrcholu.

Provedeme to jinak. Všechny aktivními vrcholy s $d(v) = k$ si uložíme do fronty D_k. Frontu realizujeme jako obousměrný spojový seznam. Ten umožní provádět vkládání a mazání v konstantním čase. Navíc si pro každý vrchol budeme pamatovat ukazatel, který nám umožní přístup k položce ve správné frontě v konstantním čase (pro aktivní vrcholy).

Po provedení zvýšení vrcholu jednoduše přesuneme vrchol do jiné fronty. Pokud protlačení toku po hraně vw aktivuje w, tak ho vložíme do správné fronty. Pokud protlačení deaktivuje vrchol v, tak ho vyřadíme z fronty. Tyto operace proběhnou v konstantním čase.

Proč je tak jednoduché najít aktivní vrchol s maximálním $d(v)$? Po zvýšení v zůstane vrchol v aktivní a s maximálním $d(v)$. Dejme tomu, že $d(v) = k$. Pokud protlačení po vw deaktivuje v, tak se nejprve podíváme do fronty D_k. Když je prázdná, tak se podíváme do fronty D_{k-1}. Skoro vždy v ní najdeme aktivní vrchol, protože poslední protlačení po vw splňovalo $d(w) = k-1$. Vrchol w musí být aktivní, jinak je $w = s$ nebo $w = t$.

Případ $w = t$ je triviální, protože potom není žádný vrchol aktivní a algoritmus skončí.

Jediný případ, kdy neuspějeme s hledáním aktivního vrcholu, nastane, když zpracujeme vrchol z D_{n+1} a obě fronty D_{n+1}, D_n jsou prázdné. Než tato situace nastane znova, tak se bude muset nějaký vrchol dostat do fronty D_{n+1}. Neboli jeho $d(v)$ bude muset překročit n. To může nastat pro každý vrchol nejvýše jednou a proto tento špatný případ nastane nejvýše n krát. V tomto špatném případě budeme muset prohledat všechny fronty D_k s $k < n$. Špatné případy celkem přispějí do hledání aktivních vrcholů časem $\mathcal{O}(n^2)$. Celkem hledání aktivních vrcholů zabere čas $\mathcal{O}(N + n^2)$, kde N je počet všech protlačení.

Ukázali jsme si, že maximum distance push-relabel algoritmus může být implementován tak, aby běžel v čase $\mathcal{O}(n^3)$. Pokud bychom použili odhad $\mathcal{O}(n^2\sqrt{m})$ na počet protlačení, tak dokonce v čase $\mathcal{O}(n^2\sqrt{m})$.

Heuristika zrychlující Push-Relabel algoritmus

Ještě zmíníme heuristiku, které nemá vliv na odhad časové složitosti, ale v praxi podstatně zrychlí výpočet. Pravidelně, řekněme po $n/2$ zpracováních vrcholů, přepočítáme platné označkování. Ukazovali jsme si, že platné označkování $d(v)$ je dolním odhadem na $d_f(v,t)$ a že $d(v) - n$ je dolním odhadem $d_f(v,s)$. Zvolme proto nové označkování jako $d(v) := \min\{d_f(v,t),\ n + d_f(v,s)\}$. Ukažte, že toto označkování je platné a v jistém smyslu nejlepší možné. Také si rozmyslete, jak rychle ho můžeme spočítat.

Poznámka: Předvýpočet platného označkování se vyplatí už při inicializaci push-relabel algoritmu.

1.4 Srovnání algoritmů pro hledání maximálního toku

Přehled časových složitostí variant algoritmu vylepšující cesty.

Dinic	$\mathcal{O}(n^2m)$
3 indové	$\mathcal{O}(n^3)$
nejlepší známé	$\mathcal{O}(mn\log(n^2/m))$
jednotkové kapacity	$\mathcal{O}(\sqrt{m}\cdot m)$
jednotkové kapacity, prostý graf	$\mathcal{O}(n^{2/3}\cdot m)$
jednotkové kapacity, vstupní nebo výstupní stupeň ≤ 1	$\mathcal{O}(\sqrt{n}\cdot m)$
celočíselné kapacity	$\mathcal{O}(\lvert f\rvert \cdot n + nm)$
celočíselné kapacity $\leq U$	$\mathcal{O}(Un^2 + nm)$
scaling (celočíselné kapacity $\leq U$)	$\mathcal{O}(nm\log U)$

Prostým grafem myslíme graf bez násobných hran.[13] Ostatní algoritmy fungují i na multigrafech.

Golgberg a Tarjan [7] přišli s algoritmem na nalezní blokujícího toku v acyklickém orientovaném grafu v čase $\mathcal{O}(m\log(n^2/m))$. Důsledkem toho dostáváme algoritmus pro hledání maximálního toku v obecných orientovaných grafech v čase $\mathcal{O}(nm\log(n^2/m))$.

Většinu algoritmů pro speciální případy naleznete ve cvičeních. Podle navrhnutých základních myšlenek nebo kostry algoritmu si sami zkusíte domyslet detaily, sestavit algoritmus a dokázat, že funguje. Upočítání časových složitostí pro speciální případy naleznete v [9].

Přehled časových složitostí variant push-relabel algoritmu.

maximum distance push-relabel	$\mathcal{O}(n^2\sqrt{m})$
jednotkové kapacity	$\mathcal{O}(nm)$
celočíselné kapacity $\leq k$	$\mathcal{O}(knm)$
scaling přebytků	$\mathcal{O}(nm + n^2\log U)$

S některými variantami se opět seznámíte ve cvičeních.

Scaling přebytků je varianta push-relabel algoritmu, která protlačuje tok z vrcholů s dostatečně vysokým přebytkem do vrcholů s dostatečně nízkým přebytkem, přičemž nikdy nedovolíme, aby byl přebytek příliš velký. Myšlenka scalingu je podobná scalingu u algoritmů vylepšující cesty. Celkem hezky je to popsané v [2].

Poznamenejme, že push-relabel algoritmus se chová dobře i na speciálních grafech. Například na bipartitním grafu s n_1 a n_2 vrcholy doplněném o zdroj a spotřebič běží FIFO push-relabel algoritmus v čase $\mathcal{O}(n_1m + n_1^3)$ (podívejte se do [3]).

Nejlepší známé řešení pro toky v sítích používá nový přístup k problému. Je to algoritmus Golberg-Rao [5] s časovou složitostí $\mathcal{O}(m\sqrt{m}\log(n^2/m)\log U)$.[14]

Praktické chování.

Poznamenejme, že algoritmus push-relabel se v praxi chová velice dobře a je podstatně rychlejší než algoritmy vylepšující cesty. (Můžeme si to vysvětlit tím, že pomocí platného označkování snadněji „udržujeme“ vrstevnatou síť a nemusíme ji pokaždé přepočítávat.)

Push-relabel algoritmus můžeme ještě více zrychlit pomocí heuristiky popsané v podsekci 1.3.

[13] Tedy ne multigraf. Ikdyž pokud je násobnost hrany omezena pevnou konstantou, tak to nevadí.

[14] Dokonce se dá ukázat, že člen $\sqrt{m}$ v časové složitosti může být nahrazen členem $\min\{\sqrt{m}, n^{2/3}\}$.

1.5 Aplikace toků v sítích

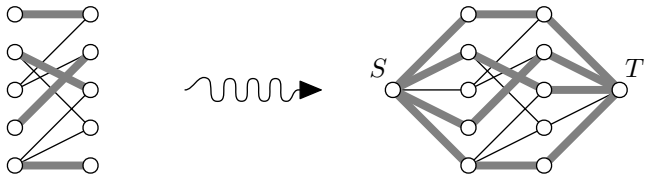
Pomocí toků v sítích se dá vyřešit nepřeberné množství problémů. Představíme si několik vzorových aplikací a ostaní si necháme jako cvičení.

1.5.1 Maximální párování v bipartitním grafu

Definice: Množina hran $M \subseteq E$ v grafu $G = (V, E)$ je *párování* pokud žádný vrchol $v \in V$ neleží ve dvou hranách M. Jinými slovy, párování je množina nezávislých hran. Párování M je *maximální*, pokud pro každé párování M' platí $|M| \geq |M'|$.

Věta 4 *Maximální párování v bipartitním grafu $G = (V_1 \cup V_2, E)$ můžeme spočítat pomocí toku v síti v čase $\mathcal{O}(n^2 m)$.*

Důkaz: Zkonstruujeme orientovaný graf $G' = (V_1 \cup V_2 \cup \{S, T\}, E')$ následujícím způsobem. K původnímu grafu G přidáme super-zdroj S a super-spotřebič T. Super-zdroj spojíme se všemi vrcholy $v_1 \in V_1$ hranou Sv_1. Každý vrchol $v_2 \in V_2$ spojíme se super-spotřebičem hranou v_2T. Původní hrany grafu orientujeme z množiny V_1 do V_2. Kapacity všech hran nastavíme na 1.



Tvrdíme, že v grafu G' existuje maximální tok velikosti k právě tehdy, když v bipartitním grafu G existuje maximální párování velikosti k.

Nejprve ukažme první implikaci. Protože jsou kapacity hran v grafu G' celočíselné, tak v G' existuje celočíselný maximální tok (důsledek 3). Tok po každé hraně je buď 0 nebo 1. Protože do každého vrcholu $v \in V_1$ může přitékat nejvýše tok velikosti 1 a z každého vrcholu $v \in V_2$ může odtékat nejvýše tok velikosti 1, tak je maximální celočíselný tok v G' sjednocením k vrcholově disjunktních[15] (S, T)-cest. Ty obsahují disjunktní hrany tvořící párování.

Na druhou stranu můžeme hrany tvořící párování v G rozšířit do vrcholově disjunktních (S, T)-cest grafu G'.

Pro nalezení maximálního párování v bipartiním grafu G tedy stačí najít maximální tok v pomocném grafu G'. ∎

Pomocný graf G' má speciální tvar a dá se ukázat, že v něm Dinicův algoritmus trvá jen čas $\mathcal{O}(n^{2/3}m)$ (viz cvičení).[16] Takže maximální párování ve skutečnosti najdeme rychleji.

1.5.2 Cirkulace s požadavky

Malinko zobecníme problém toků v sítích. Nechť $G = (V, E)$ je orientovaný graf. Každá hrana $e \in E$ má kapacitu $c(e)$, kde $c : E \to \mathbb{R}_+$. Každý vrchol $v \in V$ má požadavek $d(v) \in \mathbb{R}$. Chceme, aby ve vrcholu v zůstal přebytek toku $d(v)$. Vrchol v s $d(v) > 0$ se chová jako spotřebič, vrchol v s $d(v) < 0$ jako zdroj a vrchol v s $d(v) = 0$ jako normální vrchol.

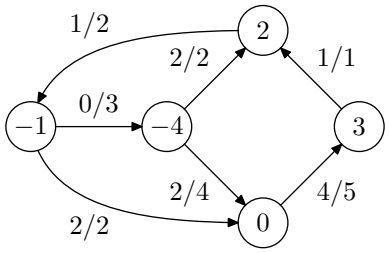
Definice: *Cirkulace s požadavky* $\{d(v)\}_{v \in V}$ je funkce $f : E \to \mathbb{R}_+$, která splňuje

[15] Dvě cesty jsou vrcholově disjunktní, pokud nemají společný vrchol – kromě počátku a konce, kde se to toleruje.

[16] Poznamenejme, že hledání maximálního párování v bipartitním grafu pomocí Dinicova algoritmu je ekvivalentní Hopcroft-Karpovu algoritmu využívajícího volné střídavé cesty.

$$\begin{array}{lll} i) & 0 \le f(e) \le c(e) & \text{pro každou hranu } e \in E \\ ii) & f(v) = d(v) & \text{pro každý vrchol } v \in V \end{array}$$

Problém: Existuje v síti G cirkulace splňující požadavky?



Na obrázku je příklad cirkulace f v grafu G. Hodnoty ve vrcholech označují požadavky $d(v)$ a hrany jsou označeny „$f(e)/c(e)$“.

Pozorování 2 *Pokud v síti G existuje cirkulace splňující požadavky $\{d(v)\}_{v\in V}$, tak je $\sum_{v\in V} d(v) = 0$.*

Důkaz: Použijeme počítání dvěma způsoby. Na jednu stranu je $X = \sum_{v\in V} d(v)$. To jsme sečetli příspěvky přebytků po vrcholech. Na druhou stranu můžeme příspěvky do X počítat po hranách.[17] Tok po každé hraně přispívá do X dvakrát (za každý konec hrany). Jednou s kladným a podruhé se záporným znaménkem. Proto je $X = 0$. ■

Pozorování zároveň říká, že pokud existuje cirkulace f, tak je

$$|f| = \sum_{v\in V, d(v)>0} d(v) = \sum_{v\in V, d(v)<0} -d(v).$$

Jak tedy zjistit, zda v G existuje cirkulace splňující požadavky? Nejprve zkontrolujeme, jestli $\sum_{v\in V,d(v)>0} d(v) = -\sum_{v\in V,d(v)<0} d(v)$. Pokud tato nutná podmínka platí, tak zkonstruujeme pomocný graf $G' = (V \cup \{S,T\}, E')$.

- Vytvoříme super-zdroj S a spojíme ho se všemi vrcholy v, které mají $d(v) < 0$ (chovají se jako „zdroje“). Kapacitu hrany Sv nastavíme na $-d(v)$.
- Vytvoříme super-spotřebič T a spojíme ho se všemi vrcholy v, které mají $d(v) > 0$ (chovají se jako „spotřebiče“). Kapacitu hrany vT nastavíme na $d(v)$.

Ve výsledné síti (která už má jen jeden zdroj a jeden spotřebič) nalezneme maximální tok. Pokud je jeho velikost rovna $\sum_{v\in V,d(v)>0} d(v)$, tak je restrikce[18] nalezeného toku na původní graf platnou cirkulací. Pokud je velikost maximálního toku menší, tak platná cirkulace neexistuje (zkuste si to dokázat).

Jako důsledek poznatků o tocích v sítích dostáváme, že když jsou všechny kapacity hran a pořadavky toku ve vrcholech celočíselné, tak existuje platná cirkulace, která je celočíselná.

[17] Připomeňme, že $d(v) = f(v) = \sum_{e=xv} f(e) - \sum_{e=vx} f(e)$.

[18] Restrikce je omezení se na určitou část. V tomto případě necháme ve funkci f pouze hrany původního grafu.

1.5.3 Cirkulace s limity na průtok hranou

V předchozí podsekci jsme si vysvětlili, co je to cirkulace v grafu $G = (V, E)$ s požadavky $\{d(v)\}_{v \in V}$. Každá hrana sítě $e \in E$ má svojí kapacitu $c(e)$, která je horním limitem na velikost toku po hraně. Tentokrát chceme velikost toku po hraně omezit i ze spoda. Hodnota $\ell(e)$ určuje minimální tok po hraně e.

Chceme nalézt tok f, který splňuje požadavky $d(v)$ ve vrcholech a navíc $\ell(e) \leq f(e) \leq c(e)$ pro každou hranu $e \in E$. Jak takový tok najít?

Zkusme následující, na první pohled naivní přístup. Na začátku po každé hraně e pošleme přesně $f_0(e) := \ell(e)$. Funkce f_0 splňuje omezení na průtok po hranách. Jediné, co brání funkci f_0 v tom, aby byla cirkulací s požadavky, jsou přebytky toku ve vrcholech. Přebytek toku ve vrcholu v je $f_0(v)$ a my potřebujeme, aby byl $d(v)$. Pokud $f_0(v) = d(v)$, tak je požadavek pro vrchol v splněn. V opačném případě musíme „tok“ f_0 upravit.

Provedeme to následovně. Nechť G' je graf G, ve kterém zvolíme nové požadavky $d'(v) := d(v) - f_0(v)$ pro každý vrchol $v \in V$ a nastavíme kapacity hran na $c'(e) := c(e) - l(e)$. Jinými slovy, od kapacit hran a požadavků ve vrcholech odečteme nutný minimální tok po hranách a dostaneme síť G'.

Všimněme si, že v G' „zmizely“ požadavky na minimální tok po hranách (požadavek $f(e) \geq l(e)$ se změnil na $f(e) \geq 0$). Pokud v G' najdeme cirkulaci f' splňující nové požadavky, tak tok $f := f' + f_0$ bude platnou cirkulací v G splňující požadavky ve vrcholech i limity na průtoky po hranách.

Tím jsme problém cirkulace s limity na průtok hranou převedli na předchozí případ „obyčejné“ cirkulace s požadavky, který umíme vyřešit pomocí hledání „klasického“ maximálního toku.

Pro cirkulace s limity na průtok hranou opět platí důsledek, že pokud jsou všechny požadavky ve vrcholech, dolní i horní limity na průtok hranou celočíselné, tak existuje cirkulace splňující požadavky a limity, která je celočíselná.

1.5.4 Rozvrhování letadel

Problém: Letecká společnost zajišťuje několik pravidelných linek mezi evropskými městy. Dostanete detailní informace o množině letů $\mathcal{L}$. Zajímalo by nás, kolik nejméně letadel je potřeba k zajištění všech letů $\mathcal{L}$. Příklad leteckého řádu je v následující tabulce.

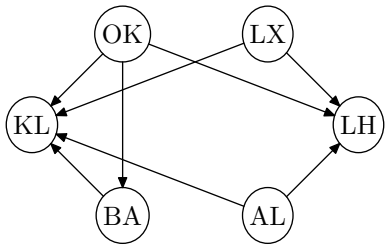
číslo letu	počátek	cíl
OK652	Praha (6am)	Londýn (8am)
LX2008	Milano (7am)	Vídeň (10am)
BA101	Londýn (9am)	Madrid (11am)
AL504	Milano (10am)	Franfurt (11am)
LH2451	Frankfurt (1pm)	Budapešť (3pm)
KL404	Barcelona (6pm)	Budapešť (9pm)

Lety i a j mohou být obslouženy stejným letadlem pokud je cílové letiště i stejné jako počáteční letiště j, a pokud je mezi oběma lety dostatek času na provedení údržby (úklid, doplnění paliva, apod). Další možností je, že letadlo přeletí z cílového letiště i na počáteční letiště j. To ovšem zabere více času a prostoj mezi oběma lety musí být výrazně delší. Přelety stojí výrazně méně než koupě dalšího letadla. Následující tabulka obsahuje příklad 3 letů, které mohou být obslouženy stejným letadlem.

číslo letu	počátek	cíl
OK652	Praha (6am)	Londýn (8am)
BA101	Londýn (9am)	Madrid (11am)
KL404	Barcelona (6pm)	Budapešť (9pm)

Modelování problému: Závislosti mezi jednotlivými lety budeme modelovat orientovaným grafem G. Lety budou tvořit vrcholy grafu a mezi dvěma vrcholy i a j povede orientovaná hrana, pokud letadlo obsluhující let i může obsloužit i let j. Orientované hrany dodržují časovou souslednost. Hrana ij mimo jiné znamená, že let i předchází letu j. Orientovaný graf G je acyklický, protože například časy odletu jednotlivých letů určují topologické uspořádání vrcholů.

Následující obrázek zachycuje modelový graf G pro množinu letů $\mathcal{L}$ z předchozího příkladu.



Otázka, jestli množina letů $\mathcal{L}$ jde obsloužit pomocí k letadel, odpovídá otázce, jestli v grafu G existuje k vrcholově disjunktních cest pokrývajících všechny vrcholy (cesta pokrývá vrchol, pokud přes něj vede).[19]

Řešení: Chceme zjistit, jestli v modelovém grafu G existuje nejvýše k vrcholově disjunktních cest, které pokryjí všechny vrcholy. Problém převedeme na výpočet cirkulace. Hlavní myšlenkou je, že podél každé cesty P_α odpovídající přeletům letadla α pošleme jednotkový tok. Pomocný graf H zkonstruujeme následovně:

- Pro každý let i vytvoříme dva vrcholy $u_i, v_i \in V(H)$. První odpovídá odletu a druhý příletu. Do grafu H ještě přidáme zdroj s a stok t. Požadavky vrcholů nastavíme na $d(s) = -k$, $d(t) = k$ a $d(x) = 0$ pro všechny ostatní vrcholy $x \in V(H)$.

- Každý let i musí být obsloužen. Proto mezi u_i a v_i přidáme hranu a nastavíme její horní i dolní limit na tok na 1. Tedy $l(u_iv_i) = 1$ a $c(u_iv_i) = 1$.

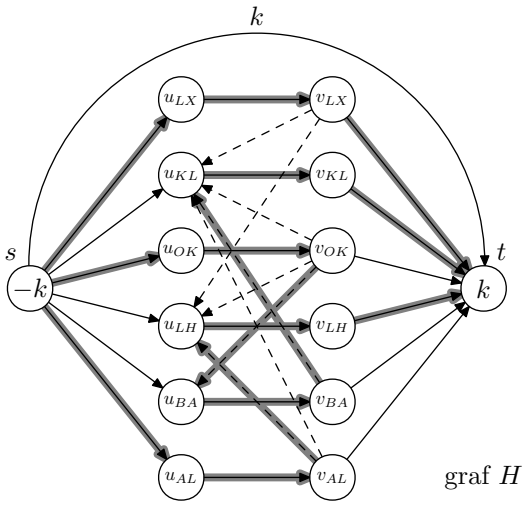
- Pokud může být let i a posléze i let j obsloužen stejným letadlem, tak do H přidáme hranu v_iu_j a nastvíme její horní limit na tok 1. Dolní limit necháme nenastaven. Tedy $c(v_iu_j) = 1$ a $l(v_iu_j) = 0$.

- Protože každé letadlo může zahájit letový den letem i, tak do H přidáme hranu su_i s horním limitem 1 a dolním 0.

- Podobně každé letadlo může skončit den letem j a proto do H přidáme hranu v_jt s horním limitem 1 a dolním 0.

- Může se stát, že nám na obsloužení všech letů bude stačit méně letadel. Proto přidáme hranu st s horním limitem k a dolním 0, po které přebytečná letadla přetečou z s do t.

[19] Porývání grafu cestami si můžeme představit tak, že se cesta potáhne asfaltem. Pak jsou všechny vrcholy/křižovatky na cestě doslova pokryty.



Lemma 14 *Lety $\mathcal{L}$ lze obsloužit pomocí k letadel právě tehdy pokud v pomocném grafu H existuje cirkulace.*

Důkaz: Předpokládejme, že lety $\mathcal{L}$ lze obsloužit pomocí $k' \leq k$ letadel. Nechť $\mathcal{L}(\alpha)$ jsou lety přiřazené letadlu α. Rozvrh letadla α odpovídá orientované cestě P_α v grafu H, která začíná v s, prochází hrany $u_i v_i$ obsluhovaných letů $i \in \mathcal{L}(\alpha)$ a končí v t, Podél této cesty P_α pošleme jednotkový tok. Pokud letadlu α nebyl přiřazen žádný let, tak pošleme jednotkový tok z s rovnou do t po hraně st. Požadavky ve vrcholech grafu H jsou splněny, protože máme k letadel a každé letadlo začíná v s a končí v t. Limity na tok po hranách jsou rovněž splněny, protože každý let byl obsloužen nějakým letadlem.

Pro důkaz druhé implikace uvažme přípustnou cirkulaci v grafu H. Protože všechny limity na hranách jsou celočíselné, tak existuje přípustná cirkulace, která je celočíselná. Tok po každé hraně kromě hrany st je buď 0 a nebo 1. Tok po hraně st může být až k. Jediný vrchol, který se chová jako zdroj, je s. Podobně jediný stokový vrchol je t. Z toho důvodu můžeme cirkulaci rozdělit na k jednotkových toků. Ty odpovídají k dijsjunktním (s,t)-cestám v H. Výjimkou jsou cesty po hraně st. Protože hrana $u_i v_i$ má horní i dolní limit 1, tak každá hrana $u_i v_i$ leží v právě jedné cestě. Každý jednotkový tok odpovídá jednomu letadlu a hrany $u_i v_i$ přes které tok prochází určují lety, které letadlu přiřadíme. ∎

Poznámka: O proti reálnýmu rozvrhování letadel jsme si problém docela zjednodušili. (i) Ve skutečnosti nestačí rozvrhovat pouze letadla, ale každému letadlu musíme přiřadit posádku. Posádka přináší další omezení, protože musí dodržovat pravidelný odpočinek (to letadla nemusí). (ii) Také bychom chtěli maximalizovat zisk. Proto bychom se chtěli vyhnout některým přeletům prázdných letadel. Každé hraně grafu H přiřadíme cenu přeletu a budeme hledat cirkulaci, která minimalizuje celkovou cenu hran, po kterých něco teče. To vede na problém toků minimální ceny (mincost flows), které se „naštěstí“ umí také dobře počítat (bližší informace čtenář najde v [4]).

Poznámka: Úloha se dá vyřešit i bez cirkulací. Můžeme ji přímo převést na toky v sítích – viz cvičení 11 v sekci 1.6.4.

1.6 Příklady

1.6.1 Toky a řezy

1. Nechť $G = (V, E)$ je síť a f_1, f_2 jsou toky v G (funkce z $E \to \mathbb{R}_+$). Rozhodněte jestli jsou následující funkce také tokem v G.

   - $f_1 + f_2$.
   - αf_1 pro $\alpha \geq 0$.

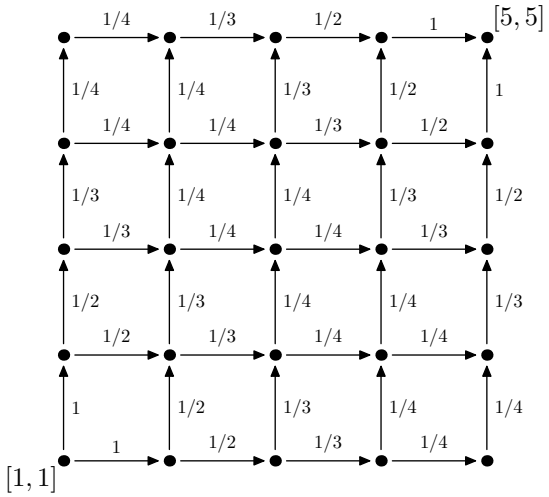
   Funkce $f_1 + f_2$ je definována jako $(f_1 + f_2)(e) := f_1(e) + f_2(e)$ pro každou hranu $e \in E$ (sčítání po složkách). Podobně $(\alpha f_1)(e) := \alpha f_1(e)$ pro každou hranu $e \in E$.

   Jaké vlastnosti (podmínky) z definice toku mohou být porušeny? Co musí funkce $f_1 + f_2$, respektive αf_1, splňovat, aby byla tokem?

2. Nechť $G = (V, E)$ je síť tvaru mřížky 5×5 s vrcholy $V = \{[x, y] : 1 \leq x, y \leq 5\}$ a orientovanými hranami $([x, y], [x + 1, y])$ a $([x, y], [x, y + 1])$ s kapacitami

$$c([x, y][u, v]) = \frac{1}{\min\{x + y - 1,\ 10 - x - y\}}.$$

   Určete maximální tok ze zdroje $[1, 1]$ do stoku $[5, 5]$.

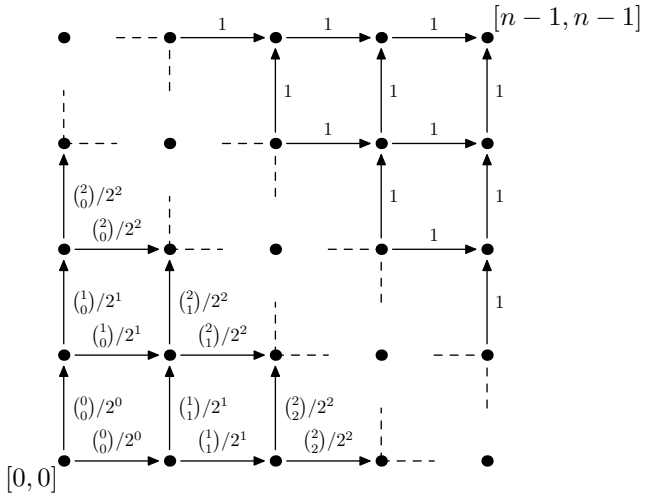


   Nápověda: $\frac{5}{3}$.

3. Nechť $G = (V, E)$ je síť tvaru mřížky $n \times n$ s vrcholy $V = \{[x, y] : 0 \leq x, y \leq n - 1\}$ a orientovanými hranami $([x, y], [x + 1, y])$ a $([x, y], [x, y + 1])$ s kapacitami

$$c([x, y][u, v]) = \begin{cases} \binom{x+y}{x}/2^{x+y} & \text{pro } x + y \leq n - 2 \\ 1 & \text{jinak.} \end{cases}$$

Určete maximální tok ze zdroje $[0,0]$ do stoku $[n-1,n-1]$.



Nápověda: 2.

4. Nechť Q_n je orientovaný graf n-dimenzionální krychle. Tj. graf jehož vrcholy jsou posloupnosti nul a jedniček délky n a dva vrcholy jsou spojeny hranou, pokud se příslušné posloupnosti liší právě v jedné souřadnici. Hranu orientujeme od posloupnosti s menším počtem jedniček do posloupnosti s větším počtem jedniček. Zvolme zdroj $z = (0,0,\ldots,0)$ a stok $s = (1,1,\ldots,1)$ a nastavme kapacity všech hran na jedničku.

   (a) Nalezněte maximální tok ze zdroje do spotřebiče.

   (b) Dokázali byste najít celočíselný tok?

   (c) Dokázali byste najít nikde nenulový tok? Tj. aby každou hranou něco teklo.

5. (Věta o minimálním toku a maximálním řezu) Nechť $G = (V,E)$ je orientovaný graf, $c : E \to \mathbb{R}_+$ je ohodnocení hran, z zdroj a s spotřebič.

   Hledáme minimální tok f ze zdroje do spotřebiče takový, aby $f(e) \geq c(e)$ pro každou hranu $e \in E$. Hodnota $c(e)$ je minimální přípustný průtok hranou. Podobně hledáme maximální řez $\delta(R)$ určený množinou $R \subseteq V$, který odděluje zdroj od spotřebiče a opačný řez $\delta(\overline{R})$ neobsahuje žádnou hranu (všechny hrany mezi R a $\overline{R}$ vedou směremem z R do $\overline{R}$).

   Dokažte, že platí

$$\max\{c(\delta(R)) : \delta(R) \ (z,s)\text{-řez takový, že } \delta(\overline{R}) = \emptyset\} =$$

$$= \min\{|f| : f \ (z,s)\text{-tok takový, že } \ \forall e \in E \ : \ f(e) \geq c(e)\}$$

   Ukažme si příklad, proč je podmínka $\delta(\overline{R}) = \emptyset$ z věty nutná. Představme si síť obsahující jen hranu zs a opačnou hranu sz, obě s ohodnocením jedna. Nejmenší přípustný tok posílá jednotku toku po jedné hraně tam a po druhé hraně zpátky. Velikost takového toku je nula. Naproti tomu velikost jediného (z,s)-řezu je jedna. V tomto případě by „věta bez podmínky“ neplatila. Žádný řez nesplňuje podmínku $\delta(\overline{R}) = \emptyset$ a proto je velikost maximálního řezu splňujícího podmínku rovna nule.[20]

[20] Pro znalce lineálního programování: Podmínka souvisí s tím, že tok je vždy nezáporný.

Jako další jednoduchý příklad si rozmyslete tok a řez v síti, kde za z připojíme ještě jednu hranu zx ohodnocenou jedničkou a prohlásíme x za nový a jediný spotřebič.

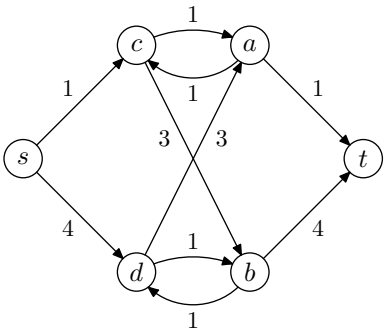
Nápověda: Postupujte podobně jako při důkazu věty o maximálním toku a minimálním řezu.

1.6.2 Algoritmy na toky v sítích

1. Nalezněte maximální tok v síti na obrázku pomocí

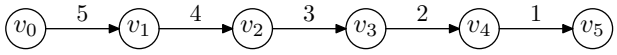
   (a) Ford-Fulkersonova algoritmu
   (b) Edmonds-Karpova algoritmu
   (c) Dinicova algoritmu
   (d) Dinicova algoritmu s metodou 3 Indů
   (e) Goldbergova Push-Relabel algoritmu.

   Který algoritmus se Vám nejjednodušeji provádí na papíře? Který algoritmus proběhne nejrychleji na počítači?



   Úlohu vyřeště ještě pro síť z příkladu 2 na straně 30. Čím je tato síť speciální?

2. Pomocí Goldbergova Push-Relabel algoritmu nalezněte maximální tok v následující síti se zdrojem v_0 a spotřebičem v_5.



   (a) Průběh celého algoritmu si odkrokujte. Kolik kroků algoritmus provede? Kolik kroků provede FIFO push-relabel algoritmus. A kolik kroků provede maximum distance push-relabel algoritmus? Výsledky porovnejte. Které pravidlo pro výběr aktivního vrcholu byste si vybrali?

   (b) Graf na obrázku je zužující cesta délky 5. Dokázali byste spočítat, kolik kroků provede která varianta push-relabel algoritmu na zužující se cestě délky n?

3. Profesor Protékal tvrdí, že ve Ford-Fulkersonově algoritmu není potřeba hledat vylepšující cesty používající hrany v protisměru. Nalezněte co nejjednoduší příklad, na kterém profesorovi dokážete, že jeho zjednodušení Ford-Fulkersonova algoritmu nenajde maximální tok.

4. (Kolik vylepšujících cest stačí?) Ukažte, že maximální tok v síti $G = (V, E)$ se dá vždy najít pomocí nejvýše $|E|$ vylepšujících cest.

   Nápověda: Vylepšující cesty určete až po nalezení maximálního toku.

5. (Kratší důkaz Edmonds-Karpova algoritmu) Dokažte, že v průběhu Edmonds-Karpova algoritmu může každá hrana zmizet ze sítě rezerv nejvýše $n/2$ krát. Z toho dále odvoďte, že Edmonds-Karpův algoritmus provede nejvýše $mn/2$ iterací (vylepšení podél nejkratší vylepšující cesty). Uměli byste ukázat ještě lepší odhad? Dokažte, že každá hrana $uv \in E$ může zmizet ze sítě rezerv nejvýše $n/4$ krát.

   Nápověda: Co víte o změnách $d_f(u)$ pro hranu $uv \in E$?

6. (Otázka k algoritmu 3 Indů) Proč musíme tok velikosti $R(v_0)$ protlačovat z vrcholu v_0 doprava a pak ještě doleva? Nemohli bychom začít ve zdroji s a protlačovat tok velikosti $R(v_0)$ pouze doprava?

7. (Jednotkové kapacity hran) Nechť $G = (V, E)$ je síť se zdrojem s, spotřebičem t a jednotkovými kapacitami hran. Předpokládejme, že G není multigraf a že ke každé hraně $uv \in E$ existuje opačná hrana $vu \in E$ (pokud ne, tak přidáme hranu s nulovou kapacitou).

   Budeme zkoumat, jak rychle poběží Dinicův algoritmus těchto sítích.

   (a) (Jednoduchý odhad) Při hledání toku v čisté síti mají všechny hrany jednotkovou rezervu. Proto při vylepšení toku podél vylepšující cesty odstraním z čisté sítě všechny hrany na této cestě. Na základě toho ukažte, že nalezení blokujícího toku v čisté síti (jedna iterace) bude trvat jen $\mathcal{O}(m)$. To potom dává časovou složitost celého algoritmu $\mathcal{O}(nm)$.

   (b) (Lepší odhad) Zastavme Dinicův algoritmus po k iteracích. Délka nejkratší cesty ze zdroje do spotřebiče je v tento moment $\ell > k$. Dosud jsme nalezli tok f_k a chtěli bychom nalézt maximální tok f. Zbývá nám tedy v síti rezerv najít tok $f_R := f - f_k$.[21] Každá vylepšující cesta zlepší tok alespoň o 1. Zbývá nám tedy najít nejvýše $|f_R|$ vylepšujících cest. Velikost toku lze ze shora odhadnout velikostí libovolného (s,t)-řezu.

   Jak najít vhodný (malý) (s,t)-řez? Zkoumejte řezy mezi jednotlivými vrstvami čisté sítě a ukažte, že existuje řez velikosti nejvýše m/k. Potom i $|f_R| \leq m/k$.

   Na základě toho odvoďte, že zbývá provédst nejvýše m/k iterací. Celkem algoritmus provede nejvýše $k + m/k$ iterací, což pro volbu $k := \sqrt{m}$ dává nejvýše $2\sqrt{m}$ iterací. Z bodu (a) víme, že jedna iterace trvá nejvýše $\mathcal{O}(m)$ a proto je časová složitost Dinicova algoritmu v této síti $\mathcal{O}(m^{3/2})$.

   (c) (Ještě lepší odhad) Myšlenka je podobná jako v předchozí části. Po k iteracích budeme chtít nalézt malý řez.

   Zkoumejte (s,t)-řezy mezi sousedními vrstvami čisté sítě a ukažte, že existuje řez velikosti nejvýše $(n/k)^2$.

   Z toho dostaneme, že algoritmus provede nejvýše $k + (k/n)^2$ iterací, což při volbě $k := n^{2/3}$ dává časovou složitost celého algoritmu $\mathcal{O}(n^{2/3}m)$.

   Nápověda: Označme s_i počet vrcholů v i-té vrstvě. Ukažte, že existuje i takové, že $s_i + s_{i+1} \leq 2n/k$. Velikost řezu mezi i-tou a $(i+1)$-ní vrstvou je rovna počtu hran mezi s_i a s_{i+1} a to je nevýše $s_i \cdot s_{i+1}$.

8. (Jednotkové kapacity a každý vrchol má vstupní nebo výstupní stupeň 1) Nechť $G = (V, E)$ je síť se zdrojem s, spotřebičem t a jednotkovými kapacitami hran. Každý vrchol má vstupní a nebo výstupní stupeň roven jedné.

   Postupujeme stejně jako v předchozím cvičení. Zastavme Dinicův algoritmus po k iteracích. Ukažte, že existuje vrstva čisté sítě, která má nejvýše n/k

[21] Pozor, tok f_R je tokem v síti rezerv a ne tokem v původní síti.

vrcholů. Z toho vyvoďte, že zbývá provést nejvýše n/k iterací a že časová složitost Dinicova algoritmu v této síti je $\mathcal{O}(\sqrt{n}m)$.

9. („Bipartitní graf“) Dostanete bipartitní graf $G = (A \cup B, E)$. Ke grafu přidáme zdroj a spojíme ho hranou se všemi vrcholy $v \in A$. Podobně přidáme spotřebič a spojíme ho hranou se všemi vrcholy $v \in B$. Všem hranám nastavíme jednotkovou kapacitu. Těmito úpravami jsme dostali síť G'. Jaká je časová složitost Dinicova algoritmu puštěného na síť G'?

   Poznámka: Jak jste se dozvěděli v sekci 1.5 o aplikacích toků v sítích, hledání maximálního toku v síti G' odpovídá hledání maximálního párování v bipartitním grafu.

   Nápověda: $\mathcal{O}(\sqrt{n}m)$.

10. (Celočíselné kapacity hran) Nechť $G = (V, E)$ je síť se zdrojem s, spotřebičem t a celočíselnými kapacitami hran $c(e) \in \{0, 1, 2, \ldots, U\}$ pro každé $e \in E$. Označme f maximální tok v síti G.

    Ukažte, že algoritmus provede nejvýše $|f|$ vylepšení podél vylepšujících cest.

    Jednu zlepšující cestu najdeme v čase $\mathcal{O}(n)$. Během celého algoritmu zabere hledání vylepšujících cest nejvýše čas $\mathcal{O}(|f|n)$. Kromě hledání vylepšujících cest ještě provádíme pročišťování čisté sítě. To během jedné iterace zabere čas $\mathcal{O}(m)$ a celkem během algoritmu zabere čas $\mathcal{O}(nm)$.

    Ukažte, že v síti G existuje (s, t)-řez velikosti nejvýše Un. Velikost libovolného toku je menší než velikost libovolného řezu. Proto je celková časová složitost Dinicova algoritmu pro sítě s celočíselnými kapacitami hran $\mathcal{O}(Un^2 + nm)$.

11. (Scaling algoritmus) Nechť $G = (V, E)$ je síť se zdrojem s, spotřebičem t a celočíselnými kapacitami hran $c(e) \in \{0, 1, 2, \ldots, U\}$ pro každé $e \in E$. Nechť $k := \lfloor \log_2 U \rfloor$, neboli k je počet bitů potřebných k zápisu největší kapacity nějaké hrany.

    (a) Dokažte, že velikost minimálního (s, t)-řezu v G je nejvýše Um. (Dokonce můžete najít (s, t)-řez velikosti nejvýše Un.)

    (b) Dostanete pevné číslo K. Ukažte, že vylepšující cesta s kapacitou alespoň K se dá v síti G nalézt v čase $\mathcal{O}(m)$, tedy pokud taková cesta existuje.

    Následující modifikace Ford-Fulkersonova algoritmu se dá použít pro hledání maximálního toku v G.

```
1: Scaling algoritmus pro maximální tok:
2:     U := max{c(e) | e ∈ E}, k := ⌊log2 U⌋
3:     f := 0 (inicializace toku na nulový tok)
4:     K := 2^k
5:     while K ≥ 1 do
6:         while existuje vylepšující cesta P kapacity alespoň K do
7:             vylepši tok f podél cesty P
8:         K := K/2
9:     return f
```

    (c) Dokažte, že předchozí scaling algoritmus vrátí maximální tok.

    (d) Ukažte, že pokaždé, když probíhá krok 5, tak je kapacita rezerv na hranách minimálního řezu G nejvýše $2Km$

    (e) Dokažte, že vnitřní cyklus na řádcích 6–7 může pro každou hodnotu K proběhnout nejvýše $\mathcal{O}(m)$ krát.

(f) Z výše dokázaných pozorování vyvoďte, že časová složitost scaling algoritmu pro hledání maximálního toku je $\mathcal{O}(m^2 \log U)$.

(g) S využitím cvičení 10 ukažte, že časová složitost scaling algoritmu je dokonce jen $\mathcal{O}(nm \log U)$.

12. (Updatování maximálního toku) Nechť $G = (V, E)$ je síť se zdrojem s, spotřebičem t a celočíselnými kapacitami hran. Předpokládejme, že už známe maximální tok v G.

    (a) Kapacita jedné hrany $uv \in E$ se zvýší o 1. Navrhněte algoritmus, který v čase $\mathcal{O}(n + m)$ přepočítá maximální tok.

    (b) Kapacita jedné hrany $uv \in E$ se sníží o 1. Navrhněte algoritmus, který v čase $\mathcal{O}(n + m)$ přepočítá maximální tok.

13. (Minimální řez pomocí Goldbergova Push-Relabel algoritmu) Předpokládejme, že už jste pomocí Goldbergova Push-Relabel algoritmu nalezli maximální tok v síti $G = (V, E)$. Navrhněte rychlý algoritmus, který nalezne minimální řez. Jak rychle poběží?

14. (Speciální případy Push-Relabel algoritmu) Analyzujte časovou složitost Golbergova Push-Relabel algoritmu v sítích s následujícími kapacitami hran. Časovou složitost vyjádřete vzhledem k n, m a k.

    (a) Všechny hrany sítě mají kapacitu 1.

    (b) Kapacity všech hran sítě leží v množině $\{1, 2, \ldots, k\}$.

    Nápověda: Kolikrát proběhne nenasycující protlačení po hraně e před tím, než se hrana e nasytí?

15. (Goldbergův Push-Relabel algoritmus) Jak by se změnila analýza obecného Golbergova push-relabel algoritmu, kdybychom vrcholy v nezvyšovali až na $\min\{d(w) \mid vw \in E(G_f)\}$ ale vždy jen o 1?

1.6.3 Modifikace sítě

V následujících úlohách vymyslete, jak upravit zadanou síť tak, abychom pro vyřešení úlohy mohli použít hledání klasického toku v síti.

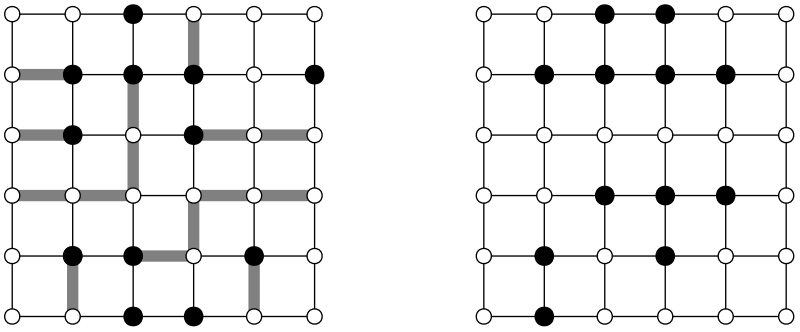
1. Co kdybychom hledali maximální tok v síti s neomezenými kapacitami hran, ale s omezením průtoku skrz vrcholy? Vyslovte a dokažte analogii Ford-Fulkersonovy věty.

2. A co kdyby byly kapacity na hranách a i ve vrcholech (omezní průtoku skrz vrchol)?

3. Jak najít maximální tok v síti, která má několik zdrojů a několik spotřebičů?

1.6.4 Aplikace toků v sítích

1. (Útěk z mřížky) Mřížka $n \times n$ je neorientovaný graf s vrcholy $[i, j]$ pro $1 \leq i, j \leq n$. Vrcholy, které se liší v právě jedné souřadnici a to o jedna, jsou spojeny hranou. Hranice mřížky obsahuje vrcholy pro které je $i = 1$, $i = n$, $j = 1$ nebo $j = n$.



Dostanete $m \leq n^2$ startovních bodů (x_1, y_1), (x_2, y_2),..., (x_m, y_m) ležících v bodech mřížky (černé body na obrázku). Problémem útěku je rozhodnout, jestli ze startovních bodů vede m vrcholově disjunktních cest na hranici mřížky (do m různých bodů na hranici mřížky). Mřížka na obrázku vlevo má řešení, mřížka na obrázku vpravo nemá.

Navrhněte efektivní algoritmus pro rozhodnutí problému útěku. Analyzujte jeho časovou složitost.

Nápověda: Síť s m zdroji a $2n - 2$ spotřebiči, kde hrany i vrcholy mají kapacitu 1. Jak tuto síť upravit, abychom mohli použít algoritmus pro hledání maximálního toku?

2. (Mengerova věta) Pomocí toků v sítích najděte:

   (a) Maximální množinu hranově disjunktních cest mezi danou dvojicí vrcholů. Jak se dá snadno ukázat, že víc cest neexistuje?

   (b) Maximální množinu vrcholově disjunktních cest mezi danou dvojicí vrcholů. Jak se dá snadno ukázat, že víc cest neexistuje?

   (c) Dokažte Mengerovu větu.
   Graf G je hranově (vrcholově) k-souvislý právě tehdy když mezi každou dvojicí vrcholů existuje k hranově (vrcholově) disjunktních cest.

3. (Königova věta) Dostenete bipartitní graf $G = (A \cup B, E)$. Pomocí toků v sítích najděte:

   (a) Maximální párování v bipartitním grafu G. Jak se dá snadno ukázat, že větší párování neexistuje? Párování $M \subseteq E$ je množina disjunktních hran (žádné dvě hrany M nemají společný vrchol).

   (b) Minimální vrcholové pokrytí v bipartitním grafu G. Vrcholové pokrytí $S \subseteq V$ je množina vrcholů taková, že pro každou hranu $e \in E$ je $S \cap e \neq \emptyset$.

   (c) Dokažte Königovu větu:
   Nechť G je bipartitní graf. Velikost maximálního párování v G je rovna velikosti minimálního vrcholového pokrytí v G.

4. (Rozvrhování letadel) V podsekci 1.5.4 jsme řešili rozvrhování letadel pomocí cirkulací. Zkuste úlohu vyřešit přímo převodem na maximální párování v bipartitním grafu.

5. (Pokrytí šachovnice dominem) Dostanete šachovnici, na které už stojí některé figurky. Rozhodněte, jestli lze všechna prázdná políčka pokrýt kostičkami 1×2? Při pokrytí se kostičky nesmí překrývat. Pokud ne, tak kolik nejvíce políček můžete pokrýt?

6. (Rozmístění věží, aby se neohrožovali) Dostanete šachovnici, která má místo některých políček díry. Kolik nejvíc věží můžete na šachovnici rozmístit tak, aby se navzájem neohrožovali? Věž se nesmí položit na díru, ale může přes ní útočit.

7. (Hallova věta) Pomocí věty o minimálním řezu a maximálním toku dokažte:

   Nechť Q je množina a $(S_1, S_2, \dots S_k)$ je systém jejích podmnožin. Systém různých reprezentantů (SRR) je množina různých prvků $\{q_1, q_2, \dots, q_k\}$ taková, že $q_i \in S_i$ pro $1 \leq i \leq k$. Hallova věta říká, že systém podmnožin má SRR právě tehdy když pro každou podmnožinu $I \subseteq \{1, 2, \dots, k\}$ platí $|\cup_{i \in I} S_i| \geq |I|$ (Hallova podmínka).

8. (Dopravní problém) Máme množinu l obchodů O a množinu k továren P. Za určité časové období je továrna i schopna vyrobit nejvýše a_i kusů produktu. Obchod j prodá za stejné časové období b_j kusů produktu. Dostaneme bipartitní graf $G = (O \cup P, E)$, kde $ij \in E$ pokud továrna i může zásobovat obchod j. Rozhodněte, jestli za zadaných podmínek dokáží továrny dostatečně zásobovat všechny obchody. Případně pro každou továrnu spočítejte, kolik kusů produktu má vyrobit a do kterých obchodů se mají produkty rozvézt.

9. (Maximalizace zisku z projektů) Jako ředitel firmy plánujete projekty na příští rok. Můžete začít realizovat projekty $P = \{p_1, p_2, \dots, p_k\}$. K realizaci některých projektů budete potřebovat některé ze zdrojů $Z = \{z_1, z_2, \dots, z_l\}$.

   Na realizaci projektu p_i vyděláte částku r_i, ale na druhou stranu k realizaci každého projektu p_i potřebujete množinu zdrojů $S_i \subseteq Z$. Každý zdroj z_j pro $1 \leq j \leq l$ je spojen s náklady c_j. Ale jakmile už zdroj z_j zakoupíme, tak ho můžeme použít ve všech projektech, které ho vyžadují.

   Navrhněte algoritmus, který zjistí, které projekty realizovat, aby byl celkový zisk co největší.

10. (Dilworthova věta) Nechť G je acyklický orientovaný graf. Dva vrcholy jsou nezávislé, pokud neexistuje orientovaná cesta z jedno vrcholu do druhého. Pokrytí grafu řetězci je množina orientovaných cest $\{P_1, P_2, \dots, P_k\}$ taková, že každý vrchol grafu leží na některé cestě P_i. Dokažte Dilworthovu věta, která říká, že velikost maximální nezávislé množiny je rovna minimálnímu počtu pokrývajících řetězců.

    Nápověda: použijte větu o minimálním toku a maximálním řezu (viz cvičení 5 v podsekci 1.6.1).

11. (Minimální pokrytí cestami) *Cestové pokrytí*[22] orientovaného grafu $G = (V, E)$ je množina $\mathcal{P}$ vrcholově disjunktních cest v G takových, že každý vrchol je obsažen právě v jedné cestě z $\mathcal{P}$. Někdy jednoduše říkáme, že každý vrchol je „pokryt“ právě jednou cestou z $\mathcal{P}$. Cesty mohou začínat a končit v libovolných vrcholech a mohou mít libovolnou délku (včetně délky 0). Minimální cestové pokrytí je cestové pokrytí nejmenším možným počtem cest.

    (a) Vymyslete efektivní algoritmus, který dostane orientovaný acyklický graf $G = (V, E)$ a najde jeho minimální cestové pokrytí.

    Nápověda: Nechť $V = \{1, 2, 3, \dots, n\}$. Zkonstruujeme orientovaný graf $G' = (V', E')$, kde $V' = \{x_0, x_1, x_2, \dots, x_n\} \cup \{y_0, y_1, y_2, \dots, y_n\}$, $E' = \{(x_0, x_i) \mid i \in V\} \cup \{(y_i, y_0) \mid i \in V\} \cup \{(x_i, y_j) \mid (i, j) \in E\}$ a pustíme na něj algoritmus pro hledání maximálního toku.

[22] z anglického „path cover“

(b) Funguje váš algoritmus i pro obecné orientované grafy? To je, nebude vadit, když G bude obsahovat orientované cykly?

12. (Orientovaný Eulerovský tah) Dostanete orientovaný graf $G = (V, E)$. *Orientovaný Eulerovský tah* je tah, který prochází každou hranu grafu právě jednou a to po směru hrany. Tah navíc skončí ve steném vrcholu, ve kterém začal (jde o orientovaný „cyklus“, který může projít některé vrcholy vícekrát).

    (a) Ukažte, že graf G obsahuje orientovaný Euleroský tah právě tehdy, když je graf G souvislý a pro každý vrchol $v \in V$ platí $deg^+(v) = deg^-(v)$ (počet šipek vstupujících do vrcholu se rovná počtu šipek vycházejících z vrcholu).

    (b) (Pošťákův problém v orientovaných grafech) Pošťák roznáší poštu ve městě, kde jsou samé jednosměrky. Mapa města odpovídá silně souvislému orientovanému grafu $G = (V, E)$ (odevšud se dá dostat kamkoliv). Pošťák potřebuje projít všechny ulice města a roznést poštu. Navrhněte mu takovou trasu, aby se nachodil co nejméně (tj. minimalizujte počet ulic, které bude muset projít vícekrát).

    Pokud v grafu nebude existovat orientovaný Eulerovský tah, tak bude pošťák muset projít některé hrany dvakrát. Navrhněte algoritmus, který rozhodne, kolik nejméně a které hrany se mají v grafu zdvojit (dostaneme tak multigraf), aby už v grafu existoval orientovaný Euleroský tah.

1.7 Doporučená literatura

Zvídavého čtenáře můžeme odkázat na přehledový článek Goldberg, Tardos a Tarjan [6] nebo knihu Ahuja, Magnanti a Orlin: *Network Flows* [1]. Mezi hezké lecture notes patří Har-Peled [8]. Z česky psaných textů můžeme doporučit Mareš: *Krajinou grafových algoritmů* [9] a Matoušek, Valla: *Kombinatorika a grafy I.* [10].

Literatura

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] R. K. Ahuja and J. B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5):748–759, 1989.

[3] R. K. Ahuja, J. B. Orlin, and C. Stein. Improved algorithms for bipartite network flow. *SIAM J. Comput.*, 23(5):906–933, 1994.

[4] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[5] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[6] A. V. Goldberg, E. Tardos, and R. E. Tarjan. Network flow algorithms. In *Paths, Flows and VLSI-Design (eds. B. Korte, L. Lovasz, H.J. Proemel, and A. Schrijver)*, pages 101–164. Springer Verlag, 1990.

[7] A. V. Goldberg and R. E. Tarjan. Solving minimum-cost flow problems by successive approximation. In *STOC*, pages 7–18. ACM, 1987.

[8] S. Har-Peled. Cs473g - graduate algorithms, 2007. `http://www.cs.uiuc.edu/class/fa07/cs473g/lectures.html`.

[9] M. Mareš. *Krajinou grafových algoritmů*. ITI Series, Prague, 2008. `http://mj.ucw.cz/vyuka/ga/`.

[10] J. Matoušek and T. Valla. Kombinatorika a grafy i., 2005. `http://kam.mff.cuni.cz/~valla/kg.html`.

[11] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer-Verlag, Berlin, 2008.