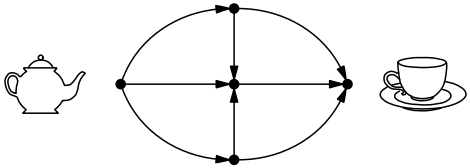
1. Toky v sítích

Už jste si někdy přáli, aby do posluchárny, kde právě sedíte, vedl čajovod a zpříjemňoval vám přednášku pravidelnými dodávkami lahodného oolongu? Nemuselo by to být komplikované: ve sklepě velikánská čajová konvice, všude po budově trubky. Tlustší od konvice do jednotlivých pater, pak by pokračovaly tenčí do jednotlivých posluchárenen. Jak ale ověřit, že potrubí má dostatečnou kapacitu na uspokojení požadavků všech čajechtivých studentů?



Obr. 1.1: Čajovod

Podívejme se na to obecněji: Máme síť trubek přepravujících nějakou tekutinu. Popíšeme ji orientovaným grafem. Jeden význačný vrchol funguje jako zdroj tekutiny, jiný jako její spotřebič. Hrany představují jednotlivé trubky, které se ve vrcholech setkávají a větví. Máme na výběr, kolik tekutiny pošleme kterou trubkou, ale přirozeně chceme ze zdroje do spotřebiče přepravit co nejvíce.

K podobné otázce dojdeme při studiu přenosu dat v počítačových sítích. Roli trubek zde hrají přenosové linky, kapacita říká, kolik dat přenesou za sekundu. Linky jsou spojené pomocí routerů a opět chceme dopravit co nejvíce dat z jednoho místa v síti na druhé. Data sice na rozdíl od čaje nejsou spojitá (přenášíme je po bytech nebo rovnou po paketech), ale při dnešních rychlostech přenosu je za taková můžeme považovat.

V této kapitole ukážeme, jak sítě a toky v nich formálně popsat, předvedeme několik algoritmů na nalezení největšího možného toku a také ukážeme, jak pomoci toků řešit jiné, zdánlivě nesouvisející úlohy.

1.1. Toky v sítích

Definice: *Síť* je uspořádaná pětice (V, E, z, s, c), kde:

- (V, E) je orientovaný graf,
- $c : E \to \mathbb{R}_0^+$ je funkce přiřazující hranám jejich *kapacity,*
- $z, s \in V$ jsou dva různé vrcholy grafu, kterým říkáme *zdroj* a *stok* (neboli *spotřebič*).

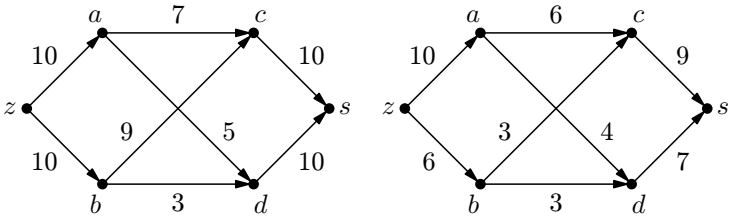
Podobně jako v předchozích kapitolách budeme počet vrcholů grafu značit n a počet hran m.

Mimo to budeme často předpokládat, že graf je symetrický: je-li uv hranou grafu, je jí i vu.[1] Činíme tak bez újmy na obecnosti: kdyby některá z opačných hran chyběla, můžeme ji přidat a přiřadit jí nulovou kapacitu.

Definice: *Tok* v síti je funkce $f : E \to \mathbb{R}_0^+$, pro níž platí:

1. Tok po každé hraně je omezen její kapacitou: $\forall e \in E : f(e) \leq c(e)$.
2. *Kirchhoffův zákon:* Do každého vrcholu přiteče stejně, jako z něj odteče („síť těsní“). Výjimku může tvořit pouze zdroj a spotřebič. Formálně:

$$\forall v \in V \setminus \{z, s\} : \sum_{u:uv \in E} f(uv) = \sum_{u:vu \in E} f(vu).$$



Obr. 1.2: Nalevo síť, napravo tok v ní o velikosti 16

Sumy podobné těm v Kirchhoffově zákoně budeme psát často, tak si pro ně zavedeme šikovné značení:

Definice: Pro libovolnou funkci $f : E \to \mathbb{R}$ definujeme:

- $f^+(v) := \sum_{u:uv \in E} f(uv)$ (celkový *přítok* do vrcholu)
- $f^-(v) := \sum_{u:vu \in E} f(vu)$ (celkový *odtok* z vrcholu)
- $f^\Delta(v) := f^+(v) - f^-(v)$ (*přebytek* ve vrcholu)

(Kirchhoffův zákon pak říká prostě to, že $f^\Delta(v) = 0$ pro všechna $v \neq z, s$.)

Definice: *Velikost toku* f označíme $|f|$. Velikost bude rovna přebytku spotřebiče $f^\Delta(s)$. Říká nám tedy, kolik tekutiny přiteče do spotřebiče a nevrátí se zpět do sítě.

Pozorování: Jelikož síť těsní, mělo by být jedno, zda velikost toku měříme u spotřebiče nebo u zdroje. Vskutku, krátkým výpočtem ověříme, že tomu tak je:

$$f^\Delta(z) + f^\Delta(s) = \sum_v f^\Delta(v) = 0.$$

[1] Nebude-li hrozit nedorozumění, budeme hranu z vrcholu u do vrcholu v značit uv namísto formálnějšího, ale méně přehledného (u, v). Podobně u neorientovaných hran napíšeme uv namísto $\{u, v\}$.

První rovnost platí proto, že podle Kirchhoffova zákona jsou zdroj a spotřebič jediné dva vrcholy, jejichž přebytek může být nenulový. Druhou rovnost získáme tak, že si uvědomíme, že tok po každé hraně přispěje do celkové sumy jednou s kladným znaménkem a jednou se záporným. Zjistili jsme tedy, že přebytek zdroje a spotřebiče se liší pouze znaménkem. □

Poznámka: Když vyslovíme nějakou definici, měli bychom se ujistit, že definovaný objekt existuje. S tokem jako takovým je to snadné: v libovolné síti splňuje definici toku všude nulová funkce. Maximální tok je zrádnější: i v jednoduché sítí najdeme nekonečně mnoho různých toků, takže není a priori jasné, že některý z nich bude maximální. Zde pomůže matematická analýza (viz cvičení 2). My si raději poradíme konstruktivně – předvedeme algoritmus, jenž maximální tok najde. Nejprve se nám to podaří pro racionální kapacity, později pro libovolné reálné.

Cvičení

1. Naše definice toku v síti úplně nepostihuje náš „čajový“ příklad z úvodu kapitoly: v něm bylo totiž spotřebičů více. Ukažte, jak tento příklad pomocí našeho modelu toků vyřešit.
2. Doplňte detaily do následujícího důkazu existence maximálního toku: Uvažme množinu všech toků coby podprostor metrického prostoru $\mathbb{R}^m$. Tato množina je omezená a uzavřená, tedy je kompaktní. Velikost toku je spojitá funkce z této množiny do $\mathbb{R}$, takže musí nabývat minima i maxima.

1.2. Fordův-Fulkersonův algoritmus

Nejjednodušší z algoritmů na hledání maximálního toku pochází od Forda a Fulkersona. Je založen na prosté myšlence: začneme s nulovým tokem a postupně ho vylepšujeme, až dostaneme maximální tok.

Uvažujme, jak by vylepšování mohlo probíhat. Nechť existuje cesta P ze z do s taková, že po všech jejích hranách teče méně, než dovolují kapacity. Takové cestě budeme říkat *zlepšující,* protože po ní můžeme tok zvětšit. Zvolme

$$\varepsilon := \min_{e \in P} \left(c(e) - f(e)\right).$$

Po každé hraně zvýšíme průtok o ε, čili definujeme nový tok f' takto:

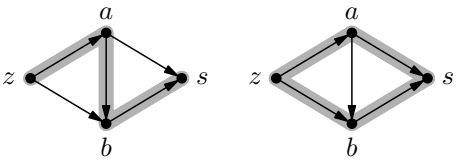
$$f'(e) := \begin{cases} f(e) + \varepsilon & \text{pro } e \in P \\ f(e) & \text{pro } e \notin P \end{cases}$$

To je opět korektní tok: kapacity nepřekročíme (ε jsme zvolili největší, pro něž se to ještě nestane) a Kirchhoffovy zákony zůstanou neporušeny, neboť zdroj a stok neomezují a každému jinému vrcholu na cestě P se zvětší o ε jak přítok $f^+(v)$, tak odtok $f^-(v)$. Také si všimneme, že velikost toku stoupla o ε.

Například v toku na obrázku 1.2 můžeme využít cestu $zbcs$ a poslat po ní 1 jednotku.

Tento postup můžeme opakovat, dokud existují nějaké zlepšující cesty, a získávat čím dál větší toky.

Až zlepšující cesty dojdou (pomiňme na chvíli, jestli se to opravdu stane), bude tok maximální? Překvapivě ne vždy. Uvažujme například síť s jednotkovými kapacitami nakreslenou na obrázku 1.3. Najdeme-li nejdříve cestu *zabs*, zlepšíme po ní tok o 1. Tím dostaneme tok z levého obrázku, ve kterém už žádná další zlepšující cesta není. Jenže jak ukazuje pravý obrázek, maximální tok má velikost 2.



Obr. 1.3: Algoritmus v úzkých (všude $c = 1$)

Tuto prekérní situaci by zachránilo, kdybychom mohli poslat tok velikosti 1 proti směru hrany ba. To nemůžeme udělat přímo, ale stejný efekt bude mít odečtení jedničky od toku po směru hrany. Rozšíříme tedy náš algoritmus, aby uměl posílat tok i proti směru hran. O kolik můžeme tok hranou zlepšit (ať už přičtením po směru nebo odečtením proti směru), to nám bude říkat její rezerva:

Definice: *Rezerva hrany* uv je $r(uv) := c(uv) - f(uv) + f(vu)$.

Definice: Hraně budeme říkat *nasycená,* pokud má nulovou rezervu. *Nenasycená cesta* je taková, jejíž všechny hrany mají nenulovou rezervu.

Budeme tedy opakovaně hledat nenasycené cesty a tok po nich zlepšovat. Tím dostaneme algoritmus, který objevili v roce 1954 Lester R. Ford a Delbert R. Fulkerson. Postupně dokážeme, že tento algoritmus je konečný a že v každé síti najde maximální tok.

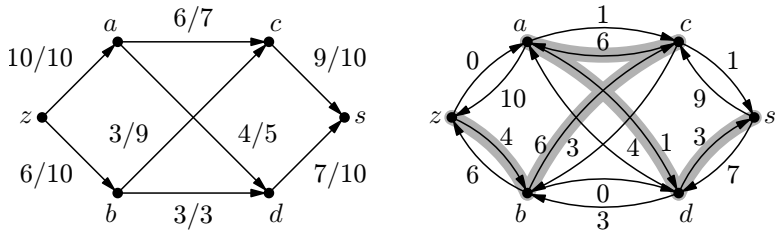
Algoritmus FordFulkerson
Vstup: Síť.

1. $f \leftarrow$ libovolný tok, např. všude nulový.
2. Dokud existuje nenasycená cesta P ze z do s, opakujeme:
3. $\quad \varepsilon \leftarrow \min\{r(e) \mid e \in P\}$.
4. $\quad$ Pro všechny hrany $uv \in P$:
5. $\quad\quad \delta \leftarrow \min\{f(vu), \varepsilon\}$
6. $\quad\quad f(vu) \leftarrow f(vu) - \delta$
7. $\quad\quad f(uv) \leftarrow f(uv) + \varepsilon - \delta$

Výstup: Maximální tok f.

Konečnost: Zastaví se Fordův-Fulkersonův algoritmus?

- Pakliže jsou všechny kapacity celá čísla, velikost toku se v každém kroku zvětší alespoň o 1. Algoritmus se tedy zastaví po nejvíce tolika



Obr. 1.4: Fordův-Fulkersonův algoritmus v řeči rezerv: vlevo tok/kapacita, vpravo rezervy a nenasycená cesta

krocích, kolik je nějaká horní mez pro velikost maximálního toku – např. součet kapacit všech hran vedoucích do stoku ($c^+(s)$).

- Pro racionální kapacity využijeme jednoduchý trik. Nechť M je nejmenší společný násobek jmenovatelů všech kapacit. Spustíme-li algoritmus na síť s kapacitami $c'(e) = c(e) \cdot M$, bude se rozhodovat stejně jako v původní síti, protože bude stále platit $f'(e) = f(e) \cdot M$. Nová síť je přitom celočíselná, takže se algoritmus jistě zastaví.
- Na síti s iracionálními kapacitami se algoritmus může chovat divoce: Nemusí se zastavit, ba ani nemusí konvergovat ke správnému výsledku (viz cvičení 2).

Maximalita: Už víme, že algoritmus se zastaví a vydá jako výsledek nějaký tok f. Je tento tok maximální? Dokážeme to pomocí řezů.

Definice: Pro libovolné dvě množiny vrcholů A a B budeme značit $E(A,B)$ množinu hran vedoucích z A do B, tedy $E(A,B) = E \cap (A \times B)$. Je-li dále f nějaká funkce přiřazující hranám čísla, označíme:

- $f(A,B) := \sum_{e \in E(A,B)} f(e)$ *(tok z A do B)*
- $f^\Delta(A,B) := f(A,B) - f(B,A)$ *(čistý tok z A do B)*

Definice: *Řez* je uspořádaná dvojice množin vrcholů (A,B) taková, že A a B jsou disjunktní, dohromady obsahují všechny vrcholy a navíc A obsahuje zdroj a B obsahuje stok. Množině A budeme říkat *levá množina řezu,* množině B *pravá.* *Kapacitu řezu* definujeme jako součet kapacit hran zleva doprava, tedy $c(A,B)$.

Lemma: Pro každý řez (A,B) a každý tok f platí $f^\Delta(A,B) = |f|$.

Důkaz: Opět šikovným sečtením přebytků vrcholů:

$$f^\Delta(A,B) = \sum_{v \in B} f^\Delta(v) = f^\Delta(s).$$

První rovnost získáme počítáním přes hrany: každá hrana vedoucí z vrcholu v B do jiného vrcholu v B k sumě přispěje jednou kladně a jednou záporně; hrany ležící celé mimo B nepřispějí vůbec; hrany s jedním koncem v B a druhým mimo přispějí

jednou, přičemž znaménko se bude lišit podle toho, který konec je v B. Druhá rovnost je snadná: všechny vrcholy v B kromě spotřebiče mají podle Kirchhoffova zákona nulový přebytek (zdroj přeci v B neleží). □

Poznámka: Původní definice velikosti toku coby přebytku spotřebiče je speciálním případem předchozího lemmatu – měří tok přes řez $(V \setminus \{s\}, \{s\})$.

Důsledek: Pro každý tok f a každý řez (A, B) platí $|f| \leq c(A, B)$. (Velikost každého toku je shora omezena kapacitou každého řezu.)

Důkaz: $|f| = f^{\Delta}(A, B) = f(A, B) - f(B, A) \leq f(A, B) \leq c(A, B)$. □

Důsledek: Pokud $|f| = c(A, B)$, pak je tok f maximální a řez (A, B) minimální. Jinými slovy pokud najdeme k nějakému toku stejně velký řez, můžeme řez použít jako certifikát maximality toku a tok jako certifikát minimality řezu. Následující lemma nám zaručí, že je to vždy možné:

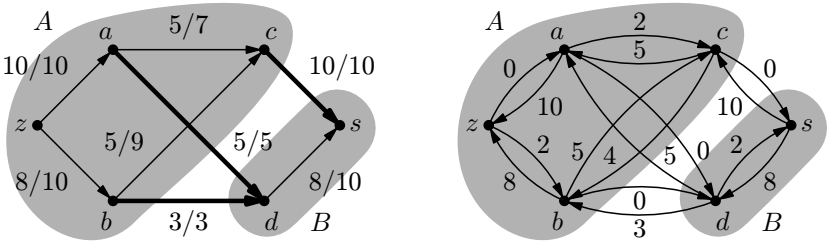
Lemma: Pokud se Fordův-Fulkersonův algoritmus zastaví, vydá maximální tok.

Důkaz: Nechť se algoritmus zastaví. Uvažme množiny vrcholů $A := \{v \in V \mid$ existuje nenasycená cesta ze z do $v\}$ a $B := V \setminus A$. Situaci sledujme na obrázku 1.5.

Všimneme si, že dvojice (A, B) je řez: Zdroj z leží v A, protože ze z do z existuje cesta nulové délky, která je tím pádem nenasycená. Spotřebič musí ležet v B, neboť jinak by existovala nenasycená cesta ze z do s, tudíž by algoritmus ještě neskončil.

Dále víme, že všechny hrany řezu mají nulovou rezervu: kdyby totiž pro nějaké $u \in A$ a $v \in B$ měla hrana uv rezervu nenulovou (nebyla nasycená), spojením nenasycené cesty ze zdroje do u s touto hranou by vznikla nenasycená cesta ze zdroje do v, takže vrchol v by také musel ležet v A, což není možné.

Proto po všech hranách řezu vedoucích z A do B teče tok rovný kapacitě těchto hran a po hranách vedoucích z B do A neteče nic. Nalezli jsme tedy řez (A, B) pro nějž $f^{\Delta}(A, B) = c(A, B)$. To znamená, že tento řez je minimální a tok f maximální. □



Obr. 1.5: Situace po zastavení F.-F. algoritmu. Nalevo tok/kapacita, napravo rezervy, v obou obrázcích vyznačen minimální řez (A, B).

Nyní již můžeme vyslovit větu o správnosti Fordova-Fulkersonova algoritmu:

Věta: Pro každou síť s racionálními kapacitami se Fordův-Fulkersonův algoritmus zastaví a vydá maximální tok a minimální řez.

Důsledek: Síť s celočíselnými kapacitami má aspoň jeden z maximálních toků celočíselný a Fordův-Fulkersonův algoritmus takový tok najde.

Důkaz: Když dostane Fordův-Fulkersonův algoritmus celočíselnou síť, najde v ní maximální tok. Tento tok bude jistě celočíselný, protože algoritmus čísla pouze sčítá, odečítá a porovnává, takže nemůže nikdy z celých čísel vytvořit necelá. □

To, že umíme najít celočíselné řešení, není vůbec samozřejmé. U mnoha problémů je racionální varianta snadná, zatímco celočíselná velmi obtížná. Více o tom řekneme v kapitole o **NP**-úplnosti. Teď si ale chvíli užívejme, že toky se v tomto ohledu chovají pěkně.

Cvičení

1. Najděte příklad sítě s nejvýše 10 vrcholy a 10 hranami, na níž Fordův-Fulkersonův algoritmus provede více než milion iterací.

2.** Najděte síť s reálnými kapacitami, na níž Fordův-Fulkersonův algoritmus nedoběhne.

3. Navrhněte algoritmus, který pro zadaný orientovaný graf a jeho vrcholy u a v nalezne největší možný systém hranově disjunktních cest z u do v.

4. Upravte algoritmus z předchozího cvičení, aby nalezené cesty byly vrcholově disjunktní (až na krajní vrcholy).

5. Jiná obvyklá definice řezu říká, že řez je množina hran grafu, po jejímž odebrání se graf rozpadne na více komponent (respektive máme-li určený zdroj a stok, skončí tyto v různých komponentách). Srovnejme tuto definici s naší. Množiny hran určené našimi řezy splňují i tuto definici a říká se jim *elementární řezy.* Ukažte, že existují i jiné než elementární řezy. Také ukažte, že všechny minimální řezy jsou elementární.

6.** Přímočará implementace Fordova-Fulkersonova algoritmu bude nejspíš graf prohledávát do šířky, takže vždy najde *nejkratší* nenasycenou cestu. Pak překvapivě platí, že algoritmus zlepší tok jen $\mathcal{O}(nm)$-krát. Návod k důkazu: Nechť $\ell(u)$ je vzdálenost ze zdroje do vrcholu u po nenasycených hranách. Nejprve si rozmyslete, že $\ell(u)$ během výpočtu nikdy neklesá. Pak dokažte, že mezi dvěma nasyceními libovolné hrany uv se musí $\ell(u)$ zvýšit. Proto každou hranu nasytíme nejvýše $\mathcal{O}(n)$-krát.

7.* Pro daný neorientovaný graf nalezněte co největší k takové, že graf je hranově k-souvislý. (To znamená, že je souvislý i po odebrání nejvýše $k-1$ hran.)

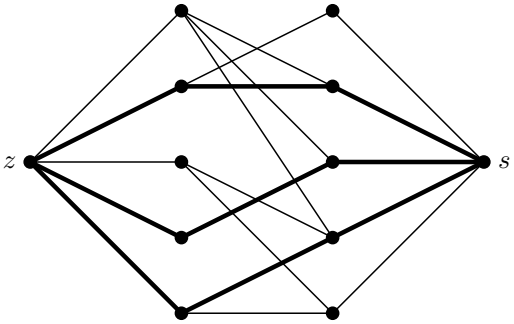
1.3. Největší párování v bipartitních grafech

Problém maximálního toku je zajímavý nejen sám o sobě, ale také na něj můžeme elegantně převádět jiné problémy. Jeden takový si ukážeme a rovnou při tom využijeme celočíselnost.

Definice: Množina hran $F \subseteq E$ se nazývá *párování*, jestliže žádné dvě hrany této množiny nemají společný vrchol. *Velikostí* párování myslíme počet jeho hran.

Chceme-li v daném bipartitním grafu (V, E) nalézt největší párování, přetvoříme graf nejprve na síť (V', E', z, s, c) takto:

- Nalezneme partity grafu, budeme jim říkat *levá* a *pravá*.
- Všechny hrany zorientujeme zleva doprava.
- Přidáme zdroj z a vedeme z něj hrany do všech vrcholů levé partity.
- Přidáme spotřebič s a vedeme do něj hrany ze všech vrcholů pravé partity.
- Všem hranám nastavíme jednotkovou kapacitu.



Obr. 1.6: Hledání největšího párování v bipartitním grafu.
Hrany jsou orientované zleva doprava a mají kapacitu 1.

Nyní v této síti najdeme maximální celočíselný tok. Jelikož všechny hrany mají kapacitu 1, musí po každé hraně téci buď 0 nebo 1. Do výsledného párování vložíme právě ty hrany původního grafu, po kterých teče 1.

Dostaneme opravdu párování? Kdybychom nedostali, znamenalo by to, že nějaké dvě vybrané hrany mají společný vrchol. Pokud by to byl vrchol pravé partity, pak do tohoto vrcholu přitekly alespoň 2 jednotky toku, jenže ty nemají kudy odtéci. Analogicky pokud by se hrany setkaly nalevo, musely by z vrcholu odtéci alespoň 2 jednotky, ale ty se tam nemají jak dostat.

Zbývá nahlédnout, že nalezené párování je největší možné. K tomu si stačí všimnout, že z toku vytvoříme párování o tolika hranách, kolik je velikost toku, a naopak z každého párování umíme vytvořit celočíselný tok odpovídající velikosti. Nalezli jsme bijekci mezi množinou všech celočíselných toků a množinou všech párování a tato bijekce zachovává velikost. Největší tok tudíž musí odpovídat největšímu párování.

Navíc dokážeme, že Fordův-Fulkersonův algoritmus na sítích tohoto druhu pracuje překvapivě rychle:

Věta: Pro síť, jejíž všechny kapacity jsou jednotkové, nalezne Fordův-Fulkersonův algoritmus maximální tok v čase $\mathcal{O}(nm)$.

Důkaz: Jedna iterace algoritmu běží v čase $\mathcal{O}(m)$: nenasycenou cestu najdeme prohledáním grafu do šířky, samotné zlepšení toku zvládneme v čase lineárním s délkou cesty. Jelikož každá iterace zlepší tok alespoň o 1, počet iterací je omezen velikostí maximálního toku, což je nejvýše n (uvažte řez okolo zdroje). □

Důsledek: Největší párování v bipartitním grafu lze nalézt v čase $\mathcal{O}(nm)$.

Důkaz: Předvedená konstrukce vytvoří z grafu síť o $n' = n + 2$ vrcholech a $m' = m + 2n$ hranách a spotřebuje na to čas $\mathcal{O}(m' + n')$. Pak nalezneme maximální celočíselný tok Fordovým-Fulkersonovým algoritmem, což trvá $\mathcal{O}(n'm')$. Nakonec tok v lineárním čase přeložíme na párování. Vše dohromady trvá $\mathcal{O}(n'm') = \mathcal{O}(nm)$. □

Cvičení

1. V rozboru Fordova-Fulkersonova algoritmu v sítích s jednotkovými kapacitami jsme použili, že tok se pokaždé zvětší alespoň o 1. Může se stát, že se zvětší víc?
2. Mějme šachovnici $R \times S$, z níž políčkožrout sežral některá políčka. Chceme na ni rozestavět co nejvíce šachových věží tak, aby se navzájem neohrožovaly. Věž můžeme postavit na libovolné nesežrané políčko a ohrožuje všechny věže v témže řádku i sloupci. Navrhněte efektivní algoritmus, který takové rozestavění najde.
3. Situace stejná jako v minulém cvičení, ale dvě věže se neohrožují přes sežraná políčka.
4. Opět šachovnice po zásahu políčkožrouta. Chceme na nesežraná políčka rozmístit kostky velikosti 1×2 políčka tak, aby každé nesežrané políčko bylo pokryto právě jednou kostkou.
5. Dopravní problém: Uvažujme továrny $T_1, \ldots, T_p$ a obchody $O_1, \ldots, O_q$. Všichni vyrábějí a prodavají tentýž druh zboží. Továrna T_i ho denně vyprodukuje t_i kusů, obchod O_j denně spotřebuje o_j kusů. Navíc známe bipartitní graf určující, která továrna může dodávat zboží kterému obchodu. Najděte efektivní algoritmus, který zjistí, zda je požadavky obchodů možné splnit, aniž by se překročily výrobní kapacity továren, a pokud je to možné, vypíše, ze které továrny se má přepravit kolik zboží do kterého obchodu.

6.** Uvažujeme o vybudování dolů $D_1, \ldots, D_p$ a továren $T_1, \ldots, T_q$. Vybudování dolu D_i stojí cenu d_i a od té doby důl zadarmo produkuje neomezené množství i-té suroviny. Továrna T_j potřebuje ke své činnosti zadanou množinu surovin (přiřazení surovin továrnám je dáno jako bipartitní graf na vstupu) a pokud jsou v provozu všechny doly produkující tyto suroviny, vyděláme na továrně zisk t_j. Vymyslete algoritmus, který spočítá, které doly postavit, abychom po odečtení nákladů na doly vydělali co nejvíce.

7.* Definujme *permament* matice $n \times n$ podobně jako determinant, jen bez znaménkového pravidla. Nahlédněte, že na permanent se dá dívat jako na součet přes všechna rozestavění n neohrožujících se věží na políčka matice, přičemž sčítáme součiny poliček, na nichž věže stojí. Jakou vlastnost bipartitního grafu

vyjadřuje permanent bipartitní matice sousednosti? ($A_{ij} = 1$, pokud vede hrana mezi i-tým vrcholem nalevo a j-tým napravo.) Radost nám kazí pouze to, že na rozdíl od determinantů neumíme permanenty počítat v polynomiálním čase.

8.* Hledání největšího párování jsme převedli na hledání maximálního toku v jisté síti. Přeložte chod Fordova-Fulkersonova algoritmu v této síti zpět do řeči párování v původním grafu. Čemu odpovídá zlepšující cesta? Získáte stejně rychlý algoritmus, který se obejde bez toků.

9.* Podobně jako v minulém cvičení přeformulujte řešení úlohy 4, aby pracovalo přímo s kostkami na šachovnici.

1.4. Dinicův algoritmus

V kapitole 1.2 jsme ukázali, jak pro nalezení maximálního toku použít Fordův-Fulkersonův algoritmus. Začali jsme s tokem nulovým a postupně jsme ho zvětšovali. Pokaždé jsme v síti našli *nenasycenou cestu*, tedy takovou, na níž mají všechny hrany kladnou rezervu. Podél takové cesty jsme pak tok zlepšili.

Ukázali jsme, že nenasycená cesta existuje právě tehdy, když tok ještě není maximální. Také jsme dokázali, že pro racionální kapacity je algoritmus konečný a vždy najde maximální tok. V obecném případě to nicméně může trvat velice dlouho.

Nyní ukážeme o něco složitější, ale výrazně rychlejší algoritmus objevený v roce 1970 Jefimem Dinicem. Jeho základní myšlenkou je nezlepšovat toky pomocí cest, ale rovnou pomocí toků ...

Síť rezerv

Nejprve přeformulujeme definici toku, aby se nám s ní lépe pracovalo. Už několikrát se nám totiž osvědčilo simulovat zvýšení průtoku hranou pomocí snížení průtoku opačnou hranou. To je přirozené, neboť přenesení x jednotek toku po hraně vu se chová stejně jako přenesení $-x$ jednotek po hraně uv.

Definice: Každé hraně uv přiřadíme její průtok $f^*(uv) = f(uv) - f(vu)$.

Pozorování: Průtoky mají následujicí vlastnosti:

(1) $f^*(uv) = -f^*(vu)$,
(2) $f^*(uv) \leq c(uv)$,
(3) $f^*(uv) \geq -c(vu)$,
(4) pro všechny vrcholy $v \neq z, s$ platí $\sum_{u:uv \in E} f^*(uv) = 0$.

Podmínka (3) přitom plyne z (1) a (2). Suma ve (4) není nic jiného než vztah pro přebytek $f^\Delta(v)$ přepsaný pomocí (1).

Lemma P: *(o průtoku)* Nechť funkce $f^* : E \to \mathbb{R}$ splňuje podmínky (1), (2) a (4). Potom existuje tok f, jehož průtokem je f^*.

Důkaz: Tok f určíme pro každou dvojici hran uv a vu zvlášť. Předpokládejme, že $f^*(uv) \geq 0$; v opačném případě u a v prohodíme. Nyní stačí položit $f(uv) := f^*(uv)$

a $f(vu) := 0$. Díky vlastnosti (2) funkce f nepřekračuje kapacity, díky (4) pro ni platí Kirchhoffův zákon. □

Důsledek: Místo toků tedy stačí uvažovat průtoky hranami. Tím se ledacos formálně zjednodušší: přebytek $f^\Delta(v)$ je prostým součtem průtoků hranami vedoucími do v, rezervu $r(uv)$ můžeme zapsat jako $c(uv) - f^*(uv)$. To nám pomůže k zobecnění zlepšujících cest z Fordova-Fulkersonova algoritmu.

Definice: *Síť rezerv* k toku f v síti $S = (V, E, z, s, c)$ je síť $R(S, f) := (V, E, z, s, r)$, kde $r(e)$ je rezerva hrany e při toku f.

Lemma Z: *(o zlepšování toků)* Pro libovolný tok f v síti S a libovolný tok g v síti $R(S, f)$ lze v čase $\mathcal{O}(m)$ nalézt tok h v síti S takový, že $|h| = |f| + |g|$.

Důkaz: Toky přímo sčítat nemůžeme, ale průtoky po jednotlivých hranách už ano. Pro každou hranu e položíme $h^*(e) := f^*(e) + g^*(e)$. Nahlédněme, že funkce h^* má všechny vlastnosti vyžadované lemmatem **P**.

(1) Jelikož první podmínka platí pro f^* i g^*, platí i pro jejich součet.
(2) Víme, že $g^*(uv) \leq r(uv) = c(uv) - f^*(uv)$, takže $h^*(uv) = f^*(uv) + g^*(uv) \leq c(uv)$.
(3) Když se sečtou průtoky, sečtou se i přebytky.

Zbývá dokázat, že se správně sečetly velikosti toků. K tomu si stačí uvědomit, že velikost toku je přebytkem spotřebiče a přebytky se sečetly. □

Poznámka: Všimněte si, že zlepšení po nenasycené cestě je speciálním případem tohoto postupu – odpovídá totiž toku v síti rezerv, který je konstantní na oné cestě a všude jinde nulový.

Dinicův algoritmus

Dinicův algoritmus bude postupně hledat nějaké pomocné toky v síti rezerv, původně nulový tok pomocí nich zlepšovat, až se dostane k maximálnímu toku. Počet potřebných iterací přitom bude záviset na tom, jak „kvalitní“ pomocné toky seženeme – na jednu stranu bychom chtěli, aby byly podobné maximálnímu toku, na druhou stranu jejich výpočtem nechceme trávit příliš mnoho času. Vhodným kompromisem jsou tzv. blokující toky:

Definice: Tok je *blokující*, jestliže na každé orientované cestě ze zdroje do spotřebiče existuje alespoň jedna hrana, na níž je tok roven kapacitě.

Blokující tok ale nebudeme hledat v celé síti rezerv, nýbrž jen v podsíti tvořené nejkratšími cestami ze zdroje do spotřebiče.

Definice: Síť je *vrstevnatá (pročištěná)*, pokud všechny její vrcholy a hrany leží na nejkratších cestách ze z do s. (Abychom vyhověli naší definici sítě, musíme ke každé takové hraně přidat hranu opačnou s nulovou kapacitou, ale ty algoritmus nebude používat a ani udržovat v paměti.)

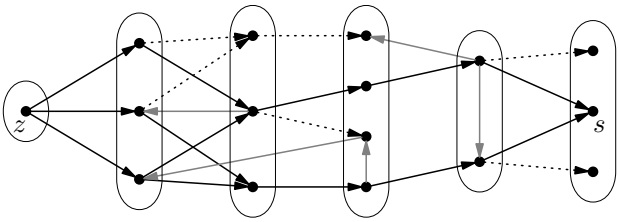
Základ Dinicova algoritmu vypadá takto:

Algoritmus DINIC
Vstup: Síť (V, E, c, z, s).

1. $f \leftarrow$ nulový tok.
2. Opakujeme:
3. Sestrojíme síť rezerv R a smažeme hrany s nulovou rezervou.
4. $\ell \leftarrow$ délka nejkratší cesty ze z do s v R.
5. Pokud $\ell = \infty$, zastavíme se a vrátíme výsledek f.
6. Pročistíme síť R.
7. $g \leftarrow$ blokující tok v R.
8. Zlepšíme tok f pomocí g.

Výstup: Maximální tok f.

Nyní je potřeba domyslet *čištění sítě.* Situaci můžeme sledovat na obrázku 1.7. Síť rozdělíme na vrstvy, přičemž v i-té vrstvě leží ty vrcholy, jejichž vzdálenost od zdroje je rovna i. Hrany vedoucí uvnitř vrstvy nebo do minulých vrstev (na obrázku šedivé) určitě neleží na nejkratších cestách. Ostatní hrany vedou o právě jednu vrstvu dopředu, ale některé z nich vedou do „slepé uličky“ (na obrázku tečkované), takže je také musíme odstranit.



Obr. 1.7: Síť rozdělená na vrstvy. Šedivé a tečkované během čištění zmizí, plné zůstanou.

Procedura ČIŠTĚNÍSÍTĚ

1. Rozdělíme vrcholy do vrstev podle vzdálenosti od z.
2. Odstraníme vrstvy za s (tedy vrcholy ve vzdálenosti větší než ℓ).
3. Odstraníme hrany do předchozích vrstev a hrany uvnitř vrstev.
4. Odstraníme „slepé uličky“, tedy vrcholy s $\deg^{\text{out}}(v) = 0$:
5. $F \leftarrow \{v \neq s \mid \deg^{\text{out}}(v) = 0\}$. *(fronta vrcholů ke smazání)*
6. Dokud $F \neq \emptyset$, opakujeme:
7. Odebereme vrchol v z F.
8. Smažeme ze sítě vrchol v i všechny hrany, které do něj vedou.
9. Pokud nějakému vrcholu klesl $\deg^{\text{out}}$ na 0, přidáme ho do F.

Nakonec doplníme *hledání blokujícího toku.* Začneme s nulovým tokem g a budeme ho postupně zlepšovat. Pokaždé najdeme nějakou orientovanou cestu ze zdroje do stoku – to se ve vrstevnaté síti dělá snadno: stačí vyrazit ze zdroje a pak vždy následovat libovolnou hranu. Až cestu najdeme, tok g podél ní zlepšíme, jak nejvíce to půjde.

Pokud nyní tok na nějakých hranách dosáhl jejich rezervy, tyto hrany smažeme. Tím jsme mohli porušit pročištěnost – pakliže nějaký vrchol přišel o poslední odchozí nebo poslední příchozí hranu. Takových vrcholů se opět pomocí fronty zbavíme a síť dočistíme. Pokračujeme zlepšováním po dalších cestách, dokud nějaké existují.[2]

Procedura BlokujícíTok
Vstup: Vrstevnatá síť R s rezervami r.

1. $g \leftarrow$ nulový tok.
2. Dokud v R existuje orientovaná cesta P ze z do s, opakujeme:
3. $\quad \varepsilon \leftarrow \min_{e \in P} (r(e) - g(e))$.
4. $\quad$ Pro všechny $e \in P$: $g(e) \leftarrow g(e) + \varepsilon$.
5. $\quad$ Pokud pro kteroukoliv e nastalo $g(e) = r(e)$, smažeme e z R.
6. $\quad$ Dočistíme síť pomocí fronty.

Výstup: Blokující tok g.

Analýza Dinicova algoritmu

Lemma K: *(o korektnosti)* Pokud se algoritmus zastaví, vydá maximální tok.

Důkaz: Z lemmatu o zlepšování toků plyne, že f je stále korektní tok. Algoritmus se zastaví tehdy, když už neexistuje cesta ze z do s po hranách s kladnou rezervou. Tehdy by se zastavil i Fordův-Fulkersonův algoritmus, a ten, jak už víme, je korektní. □

Nyní rozebereme časovou složitost. Rozdělíme si k tomu účelu algoritmus na *fáze* – tak budeme říkat jednotlivým průchodům vnějším cyklem. Také budeme předpokládat, že síť na vstupu neobsahuje izolované vrcholy, takže $\mathcal{O}(n+m) = \mathcal{O}(m)$.

Lemma S: *(o složitosti fází)* Každá fáze trvá $\mathcal{O}(nm)$.

Důkaz: Sestrojení sítě rezerv, mazání hran s nulovou rezervou, hledání nejkratší cesty i konečné zlepšování toku trvají $\mathcal{O}(m)$.

Čištění sítě (i se všemi dočišťováními během hledání blokujícího toku) pracuje taktéž v $\mathcal{O}(m)$: Smazání hrany trvá konstantní čas, smazání vrcholu po smazání všech incidentních hran taktéž. Každý vrchol i hrana jsou smazány nejvýše jednou za fázi.

Hledání blokujícího toku projde nejvýše m cest, protože pokaždé ze sítě vypadne alespoň jedna hrana (ta, na níž se v kroku 3 nabývalo minimum) a už se tam

[2] Všimněte si, že algoritmus skončí tím, že smaže všechny vrcholy i hrany. Také si všimněte, že vrcholy s nulovým vstupním stupněm jsme ani nemuseli mazat, protože se do nich algoritmus při hledání cest nikdy nedostane.

nevrátí. Nalezení cesty metodou „rovnou za nosem“ přitom trvá $\mathcal{O}(n)$. Celkem tedy $\mathcal{O}(nm)$ plus čištění, které jsme ale už započítali.

Celá jedna fáze proto doběhne v čase $\mathcal{O}(m + m + nm) = \mathcal{O}(nm)$. □

Zbývá nám jen určit, kolik proběhne fází. K tomu se bude hodit následující lemma:

Lemma C: *(o délce cest)* Délka ℓ nejkratší cesty ze z do s vypočtená v kroku 4 Dinicova algoritmu po každé fázi vzroste alespoň o 1.

Důkaz: Označme R_i síť rezerv v i-té fázi poté, co jsme z ní smazali hrany s nulovou rezervou, ale ještě před pročištěním. Nechť nejkratší cesta ze z do s v R_i je dlouhá ℓ.

Jak se liší R_{i+1} od R_i? Především jsme z každé cesty délky ℓ smazali alespoň jednu hranu: každá taková cesta totiž byla blokujícím tokem zablokována, takže alespoň jedné její hraně klesla rezerva na nulu, čímž hrana vypadla. Žádná z původních cest délky ℓ tedy již v R_{i+1} neexistuje.

To ovšem nestačí – hrany mohou také přibývat. Pokud nějaká hrana měla nulovou rezervu a během fáze jsme zvýšili tok v protisměru, rezerva se zvětšila a hrana se v R_{i+1} najednou objevila. Ukážeme ale, že všechny cesty, které tím nově vznikly, jsou příliš dlouhé.

Rozdělme vrcholy grafu do vrstev podle vzdáleností od zdroje v R_i. Tok jsme zvyšovali pouze na hranách vedoucích o jednu vrstvu dopředu, takže jediné hrany, které se mohou objevit, vedou o jednu vrstvu zpět. Ovšem každá cesta ze zdroje do spotřebiče, která se alespoň jednou vrátí o vrstvu zpět, musí mít délku alespoň $\ell + 2$ (protože spotřebič je v ℓ-té vrstvě a neexistují hrany, které by vedly o více než 1 vrstvu dopředu).

Tím je lemma dokázáno. □

Věta: Dinicův algoritmus najde maximální tok v čase $\mathcal{O}(n^2m)$.

Důkaz: Jelikož každá cesta obsahuje nejvýše n vrcholů, z lemmatu **C** plyne, že fází proběhne nejvýše n. Každá fáze podle lemmatu **S** trvá $\mathcal{O}(nm)$, což dává celkovou složitost $\mathcal{O}(n^2m)$. Speciálně se tedy algoritmus vždy zastaví, takže podle lemmatu **K** vydá maximální tok. □

Poznámka: Na rozdíl od Fordova-Fulkersonova algoritmu jsme tentokrát nikde nevyžadovali racionálnost kapacit – odhad časové složitosti se o kapacity vůbec neopírá. Nezávisle jsme tedy dokázali, že i v sítích s iracionálními kapacitami vždy existuje alespoň jeden maximální tok.

V sítích s malými celočíselnými kapacitami se navíc algoritmus chová daleko lépe, než říká náš odhad. Snadno se dá dokázat, že pro jednotkové kapacity doběhne v čase $\mathcal{O}(nm)$ (stejně jako Fordův-Fulkersonův). Uveďme bez důkazu ještě jeden silnější výsledek: v síti vzniklé při hledání největšího párování algoritmem z minulé kapitoly Dinicův algoritmus pracuje v čase $\mathcal{O}(\sqrt{n} \cdot m)$.

Cvičení

1. Dokažte, že pro jednotkové kapacity Dinicův algoritmus doběhne v čase $\mathcal{O}(nm)$.

2. Dokažte totéž pro celočíselné kapacity omezené konstantou.
3. Blokující tok lze také sestrojit pomocí prohledávání do hloubky. Pokaždé, když projdeme hranou, přepočítáme průběžné minimum. Pokud najdeme stok, vracíme se do kořene a upravujeme tok na hranách. Pokud narazíme na slepou uličku, vrátíme se o krok zpět a smažeme hranu, po níž jsme přišli. Doplňte detaily.

1.5. Goldbergův algoritmus

Představíme si ještě jeden algoritmus pro hledání maximálního toku v síti. Bude daleko jednodušší než Dinicův algoritmus z předchozí kapitoly a po pár snadných úpravách bude mít stejnou, nebo dokonce lepší časovou složitost. Jednoduchost algoritmu bude ale vykoupena trochu složitějším rozborem jeho správnosti a efektivity.

Vlny, přebytky a výšky

Předchozí algoritmy začínaly s nulovým tokem a postupně ho zlepšovaly, až se stal maximálním. Goldbergův algoritmus naproti tomu začne s ohodnocením hran, které ani nemusí být tokem, a postupně ho upravuje a zmenšuje, až se z něj stane tok, a to dokonce tok maximální.

Definice: Funkce $f : E \to \mathbb{R}_0^+$ je *vlna* v síti (V, E, z, s, c), splňuje-li obě následující podmínky:

- $\forall e \in E : f(e) \leq c(e)$ (vlna nepřekročí kapacity hran),
- $\forall v \in V \setminus \{z, s\} : f^\Delta(v) \geq 0$ (přebytek ve vrcholech je nezáporný).

Každý tok je tedy vlnou, ale opačně tomu tak být nemusí – potřebujeme se postupně zbavit nenulových přebytků ve všech vrcholech kromě zdroje a spotřebiče. K tomu nám bude sloužit následující operace:

Definice: *Převedení přebytku* po hraně uv, přičemž $f^\Delta(u) > 0$ a $r(uv) > 0$, provedeme tak, že po hraně uv pošleme $\delta = \min(f^\Delta(u), r(uv))$ jednotek toku, podobně jako v předchozích algoritmech buď přičtením po směru nebo odečtením proti směru.

Pozorování: Převedení změní přebytky a rezervy následovně:

$$
\begin{aligned}
f'^\Delta(u) &= f^\Delta(u) - \delta \\
f'^\Delta(v) &= f^\Delta(v) + \delta \\
r'(uv) &= r(uv) - \delta \\
r'(vu) &= r(vu) + \delta
\end{aligned}
$$

Rádi bychom postupným převáděním všechny přebytky buď přepravili do spotřebiče, nebo, pokud je vlna příliš velká, je přelili zpět do zdroje. Chceme se ovšem vyhnout přelévání přebytků tam a zase zpět, takže vrcholům přiřadíme *výšky* – to budou nějaká přirozená čísla $h(v)$.

Přebytek pak budeme ochotni převádět pouze z vyššího vrcholu do nižšího. Pokud se stane, že nalezneme vrchol s přebytkem, ze kterého nevede žádná nenasycená

hrana směrem dolů, budeme tento vrchol *zvedat* – tedy zvyšovat mu výšku po jedné, než se dostane dostatečně vysoko, aby z něj přebytek mohl odtéci.

Získáme tak následující algoritmus:

Algoritmus GOLDBERG
Vstup: Síť.

1. Nastavíme počáteční výšky: *(zdroj ve výšce n, ostatní ve výšce 0)*
2. $h(z) \leftarrow n$
3. $h(v) \leftarrow 0$ pro všechny $v \neq z$
4. Vytvoříme počáteční vlnu: *(všechny hrany ze z na maximum, jinde 0)*
5. $f \leftarrow$ všude nulová funkce
6. $f(zv) \leftarrow c(zv)$, kdykoliv $zv \in E$
7. Dokud $\exists u \in V \setminus \{z, s\} : f^\Delta(u) > 0$:
8. Pokud $\exists v \in V : uv \in E,\ r(uv) > 0$ a $h(u) > h(v)$, *převedeme přebytek* po hraně uv.
9. V opačném případě *zvedneme* u: $h(u) \leftarrow h(u) + 1$.

Výstup: Maximální tok f.

Analýza algoritmu

Algoritmus je jednoduchý, ale na první pohled není vidět ani to, že se vždy zastaví, natož že by měl vydat maximální tok. Postupně o něm dokážeme několik invariantů a lemmat a pomocí nich se dobereme důkazu správnosti a časové složitosti.

Invariant A: *(základní)* V každém kroku algoritmu platí:

1. Funkce f je vlna.
2. Výška $h(v)$ žádného vrcholu v nikdy neklesá.
3. $h(z) = n$ a $h(s) = 0$.
4. $f^\Delta(s) \geq 0$.

Důkaz: Indukcí dle počtu průchodů cyklem (7. – 9. krok algoritmu):

- Po inicializaci algoritmu je vše v pořádku: přebytky všech vrcholů mimo zdroj jsou nezáporné, výšky souhlasí.
- Při převedení přebytku: Z definice převedení přímo plyne, že nepo-rušuje kapacity a nevytváří záporné přebytky. Výšky se nemění.
- Při zvednutí vrcholu: Tehdy se naopak mění jen výšky, ale pouze u vrcholů různých od zdroje a stoku. Výšky navíc pouze rostou. □

Invariant S: *(o spádu)* Neexistuje hrana uv, která by měla kladnou rezervu a spád $h(u) - h(v)$ větší než 1.

Důkaz: Indukcí dle běhu algoritmu. Na začátku mají všechny hrany ze zdroje rezervu nulovou a všechny ostatní vedou mezi vrcholy s výškou 0. V průběhu výpočtu by se tento invariant mohl pokazit pouze dvěma způsoby:

- Zvednutím vrcholu u, ze kterého vede hrana uv s kladnou rezervou a spádem 1. Tento případ nemůže nastat, neboť algoritmus by dal přednost převedení přebytku po této hraně před zvednutím.
- Zvětšením rezervy hrany se spádem větším než 1. Toto také nemůže nastat, neboť rezervu bychom mohli zvětšit jedině tak, že bychom poslali něco v protisměru – a to nesmíme, jelikož bychom převáděli přebytek z nižšího vrcholu do vyššího. □

Lemma K: *(o korektnosti)* Když se algoritmus zastaví, je f maximální tok.

Důkaz: Nejprve ukážeme, že f *je tok:* Omezení na kapacity splňuje tok stejně jako vlna, takže postačí dokázat, že platí Kirchhoffův zákon. Ten požaduje, aby přebytky ve všech vrcholech kromě zdroje a spotřebiče byly nulové. To ovšem musí být, protože nenulový přebytek by musel být kladný a algoritmus by se dosud nezastavil.

Zbývá zdůvodnit, že f *je maximální:* Pro spor předpokládejme, že tomu tak není. Ze správnosti Fordova-Fulkersonova algoritmu plyne, že tehdy musí existovat nenasycená cesta ze zdroje do stoku. Uvažme libovolnou takovou cestu. Zdroj je stále ve výšce n a spotřebič ve výšce 0 (viz invariant **A**). Tato cesta tedy překonává spád n, ale může mít nejvýše $n-1$ hran. Proto se v ní nachází alespoň jedna hrana se spádem alespoň 2. Jelikož je tato hrana součástí nenasycené cesty, musí být sama nenasycená, což je spor s invariantem **S**. Tok je tedy maximální. □

Invariant C: *(cesta do zdroje)* Mějme vrchol v, jehož přebytek $f^\Delta(v)$ je kladný. Pak existuje nenasycená cesta z tohoto vrcholu do zdroje.

Důkaz: Buď v vrchol s kladným přebytkem. Uvažme množinu

$$A := \{u \in V \mid \text{existuje nenasycená cesta z } v \text{ do } u\}.$$

Ukážeme, že tato množina obsahuje zdroj.

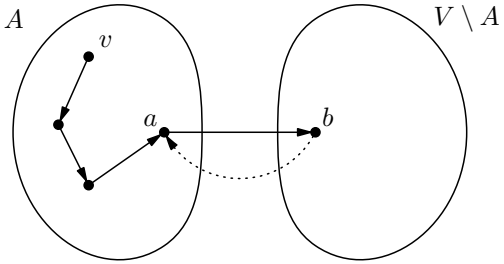
Použijeme už mírně okoukaný trik: sečteme přebytky ve všech vrcholech množiny A. Všechny hrany ležící celé uvnitř A nebo celé venku přispějí dohromady nulou. Nenulou mohou přispět pouze hrany vedoucí ven z A nebo naopak zvenku dovnitř. Získáme:

$$\sum_{u\in A} f^\Delta(u) = \underbrace{\sum_{ba\in E(V\setminus A,A)} f(ba)}_{=0} - \underbrace{\sum_{ab\in E(A,V\setminus A)} f(ab)}_{\geq 0} \leq 0.$$

Ukažme si, proč je první svorka rovna nule. Mějme hranu ab ($a \in A$, $b \in V \setminus A$). Ta musí mít nulovou rezervu – jinak by totiž i vrchol b patřil do A. Proto po hraně ba nemůže nic téci.

Druhá svorka je evidentně nezáporná, protože je to součet nezáporných ohodnocení hran.

Proto $\sum_{u\in A} f^\Delta(u) \leq 0$. Zároveň však v A leží aspoň jeden vrchol s kladným přebytkem, totiž v, tudíž v A musí být také nějaký vrchol se záporným přebytkem



Obr. 1.8: Situace v důkazu invariantu **C**

– a jediný takový je zdroj. Tím je dokázáno, že z leží v A, tedy že vede nenasycená cesta z vrcholu v do zdroje. □

Invariant V: *(o výšce)* Pro každý vrchol v je $h(v) \leq 2n$.

Důkaz: Kdyby existoval vrchol v s výškou $h(v) > 2n$, mohl se do této výšky dostat pouze zvednutím z výšky alespoň $2n$. Tehdy musel mít kladný přebytek $f^{\Delta}(v) > 0$ (jinak by nemohl být zvednut). Dle invariantu **C** musela existovat nenasycená cesta z v do zdroje. Tato cesta nicméně překonávala spád alespoň n, ale mohla mít nejvýše $n-1$ hran (na cestách se vrcholy neopakují). Tudíž musela obsahovat nenasycenou hranu se spádem alespoň 2, což je spor s invariantem **S**. □

Lemma Z: *(počet zvednutí)* Během výpočtu nastane nejvýše $2n^2$ zvednutí.

Důkaz: Z předchozího invariantu plyne, že každý vrchol mohl být zvednut nejvýše $2n$-krát. Vrcholů je n. □

Teď nám ještě zbývá určit počet provedených převedení. Bude se nám hodit, když převedení rozdělíme na dva druhy:

Definice: Řekneme, že převedení po hraně uv je *nasycené*, pokud po převodu rezerva $r(uv)$ klesla na nulu. V opačném případě je *nenasycené*, a tehdy určitě klesne přebytek $f^{\Delta}(u)$ na nulu (to se nicméně může stát i při nasyceném převedení).

Lemma S: *(nasycená převedení)* Počet všech nasycených převedení je nejvýš nm.

Důkaz: Pro každou hranu uv spočítejme počet nasycených převedení (tedy takových převedení, že po nich klesne rezerva hrany na nulu). Abychom dvakrát nasyceně převedli přebytek (nebo jeho část) z vrcholu u do vrcholu v, tak jsme museli u mezitím alespoň dvakrát zvednout:

Po prvním nasyceném převedení z vrcholu u do vrcholu v se vynulovala rezerva hrany uv. Uvědomme si, že při této operaci muselo být u výše než v, a dokonce víme, že bylo výše přesně o 1 (viz invariant **S**). Než nastane další převedení po této hraně, musíme její rezervu z nuly opět zvýšit. Jediný způsob, jak toho lze dosáhnout, je převést část přebytku z v zpátky do u. K tomu se musí v dostat (alespoň o 1) výše než u. Po přelití bude rezerva uv opět kladná. A abychom provedli nasycené převedení znovu ve směru z u do v, musíme zase u dostat (alespoň o 1) výše než v. Proto musíme u alespoň o 2 zvednout – nejprve na úroveň v a pak ještě o 1 výše.

Ukázali jsme tedy, že mezi každými dvěma nasycenými převedeními musel být vrchol u zvednut alespoň dvakrát. Podle lemmatu **V** se u mohlo zvedat maximálně $2n$-krát za celý výpočet, takže všech nasycených převedení po hraně uv je nejvýše n a po všech hranách dohromady nejvýše nm. □

Potenciálová metoda: Předchozí dvě lemmata jsme dokazovali „lokálním“ způsobem – zvednutí jsme počítali pro každý vrchol zvlášť a nasycená převedení pro každou hranu. Tento přístup pro nenasycená převedení nefunguje, jelikož jich lokálně může být velmi mnoho. Podaří se nám nicméně omezit jejich celkový počet.

Jedím ze způsobů, jak taková „globální“ tvrzení o chování algoritmů dokazovat, je použít *potenciál*. To je nějaká *nezáporná* funkce, která popisuje stav výpočtu. Pro každou operaci pak stanovíme, jaký vliv má na hodnotu potenciálu. Z toho odvodíme, že operací, které potenciál snižují, nemůže být výrazně více než těch, které ho zvyšují. Jinak by totiž potenciál musel někdy během výpočtu klesnout pod nulu.

S tímto druhem důkazu jsme se vlastně už setkali. To když jsme v kapitole o vyhledávání v textu odhadovali počet průchodů po zpětných hranách. Roli potenciálu tam hrálo číslo stavu.

V následujícím lemmatu bude potenciál trochu složitější. Zvolíme ho tak, aby operace, jejichž počty už známe (zvednutí, nasycené převedení), přispívaly nanejvýš malými kladnými čísly, a nenasycená převedení potenciál vždy snižovala.

Lemma N: *(nenasycená převedení)* Počet všech nenasycených převedení je $\mathcal{O}(n^2m)$.

Důkaz: Důkaz provedeme potenciálovou metodou. Uvažujme následující potenciál:

$$\Phi := \sum_{\substack{v \neq z,s \\ f^\Delta(v)>0}} h(v).$$

Nyní se podívejme, jak se náš potenciál během algoritmu vyvíjí:

- Na počátku je $\Phi = 0$.
- Během celého algoritmu je $\Phi \geq 0$, neboť potenciál je součtem nezáporných členů.
- Zvednutí vrcholu zvýší Φ o jedničku. (Aby byl vrchol zvednut, musel mít kladný přebytek, takže vrchol do sumy již přispíval. Teď jen přispěje číslem o 1 vyšším.) Již víme, že za celý průběh algoritmu je všech zvednutí maximálně $2n^2$, proto zvedáním vrcholů zvýšíme potenciál dohromady nejvýše o $2n^2$.
- Nasycené převedení zvýší Φ nejvýše o $2n$, protože buď po převodu hranou uv v u zůstal nějaký přebytek, takže se mohl potenciál zvýšit nejvýše o $h(v) \leq 2n$, nebo je přebytek v u po převodu nulový a potenciál se dokonce o jedna snížil. Podle lemmatu **S** nastane nejvýše nm takových nasycených převedení a ta celkově potenciál zvýší maximálně o $2n^2m$.

- Konečně když převádíme po hraně uv nenasyceně, tak od potenciálu určitě odečteme výšku vrcholu u (neboť se vynuluje přebytek ve vrcholu u) a možná přičteme výšku vrcholu v. Jenže $h(v) = h(u) - 1$, a proto nenasycené převedení potenciál vždy sníží alespoň o jedna.

Potenciál celkově stoupne o nejvyše $2n^2 + 2n^2m = \mathcal{O}(n^2m)$, klesá pouze při nenasycených převedeních a pokaždé alespoň o 1. Proto je všech nenasycených převedení $\mathcal{O}(n^2m)$. □

Implementace

Zbývá vyřešit, jak síť a výšky reprezentovat, abychom dokázali rychle hledat vrcholy s přebytkem a nenasycené hrany vedoucí s kopce.

Budeme si pamatovat seznam P všech vrcholů s kladným přebytkem. Když měníme přebytek nějakého vrcholu, můžeme tento seznam v konstantním čase aktualizovat – buďto vrchol do seznamu přidat nebo ho naopak odebrat. (K tomu se hodí, aby si vrcholy pamatovaly ukazatel na svou polohu v seznamu P). V konstantním čase také umíme odpovědět, zda existuje nějaký vrchol s přebytkem.

Dále si pro každý vrchol $u \in V$ budeme udržovat seznam $L(u)$. Ten bude obsahovat všechny nenasycené hrany, které vedou z u dolů (mají spád alespoň 1). Opět při změnách rezerv můžeme tyto seznamy v konstantním čase upravit.

Jednotlivé operace budou mít tyto složitosti:

- *Inicializace* algoritmu – triviálně $\mathcal{O}(m)$.
- *Výběr vrcholu* s kladným přebytkem a nalezení nenasycené hrany vedoucí dolů – $\mathcal{O}(1)$ (stačí se podívat na počátky příslušných seznamů).
- *Převedení přebytku* po hraně uv – změny rezerv $r(uv)$ a $r(vu)$ způsobí přepočítání seznamů $L(u)$ a $L(v)$, změny přebytků $f^\Delta(u)$ a $f^\Delta(v)$ mohou způsobit změnu v seznamu P. Vše v čase $\mathcal{O}(1)$.
- *Zvednutí vrcholu* u – musíme obejít všechny hrany do u a z u, kterých je nejvýše $2n$, porovnat výšky a případně tyto hrany uv odebrat ze seznamu $L(v)$ resp. přidat do $L(u)$. To trvá $\mathcal{O}(n)$.

Vidíme, že každé zvednutí je sice drahé, ale je jich zase poměrně málo. Naopak převádění přebytků je častá operace, takže je výhodné, že trvá konstantní čas.

Věta: Goldbergův algoritmus najde maximální tok v čase $\mathcal{O}(n^2m)$.

Důkaz: Inicializace algoritmu trvá $\mathcal{O}(m)$. Pak algoritmus provede nejvýše $2n^2$ zvednutí (viz lemma **Z**), nejvýše nm nasycených převedení (lemma **S**) a nejvýše n^2m nenasycených převedení (lemma **N**). Vynásobením složitostmi jednotlivých operací dostaneme čas $\mathcal{O}(n^3 + nm + n^2m) = \mathcal{O}(n^2m)$. Podle lemmatu **K** po zastavení vydá maximální tok. □

Cvičení

1. Rozeberte chování Goldbergova algoritmu na sítích s jednotkovými kapacitami. Bude rychlejší než ostatní algoritmy?

2. Co by se stalo, kdybychom v inicializaci algoritmu umístili zdroj do výšky $n-1$, $n-2$, anebo $n-3$?

1.6.* Vylepšení Goldbergova algoritmu

Základní verze Goldbergova algoritmu dosáhla stejné složitosti jako Dinicův algoritmus. Nyní ukážeme, že pokud budeme volit vrchol, ze kterého budeme převádět přebytek, šikovněji – totiž jako nejvyšší z vrcholů s nenulovým přebytkem –, složitost se ještě zlepší.

V časové složitosti původního algoritmu byl nejvýznamnější člen $\mathcal{O}(n^2m)$ za nenasycená převedení. Pokusme se jejich počet ve vylepšeném algoritmu odhadnout těsněji.

Lemma N': Goldbergův algoritmus s volbou nejvyššího vrcholu provede $\mathcal{O}(n^3)$ nenasycených převedení.

Důkaz: Dokazovat budeme opět pomocí potenciálové metody. Vrcholy rozdělíme do hladin podle výšky. Speciálně nás bude zajímat *nejvyšší hladina s přebytkem*:

$$H := \max\{h(v) \mid v \neq z, s \;\&\; f^{\Delta}(v) > 0\}.$$

Rozdělíme běh algoritmu na *fáze*. Každá fáze končí tím, že se H změní. Jak se může změnit? Buď se H zvýší, což znamená, že nějaký vrchol s přebytkem v nejvyšší hladině byl o 1 zvednut, nebo se H sníží. Už víme, že v průběhu výpočtu nastane $\mathcal{O}(n^2)$ zvednutí, což shora omezuje počet zvýšení H. Zároveň si můžeme uvědomit, že H je nezáporný potenciál a snižuje se i zvyšuje přesně o 1. Počet snížení bude proto omezen počtem zvýšení. Tím pádem nastane všeho všudy $\mathcal{O}(n^2)$ fází.

Během jedné fáze přitom provedeme nejvýše jedno nenasycené převedení z každého vrcholu. Po každém nenasyceném převedení po hraně uv se totiž vynuluje přebytek v u a aby se provedlo další nenasycené převedení z vrcholu u, muselo by nejdříve být co převádět. Muselo by tedy do u něco přitéci. My ale víme, že převádíme pouze shora dolů a u je v nejvyšší hladině (to zajistí právě ono vylepšení algoritmu), tedy nejdříve by musel být nějaký jiný vrchol zvednut. Tím by se ale změnilo H a skončila by tato fáze.

Proto počet všech nenasycených převedení během jedné fáze je nejvýše n. A již jsme dokázali, že fází je $\mathcal{O}(n^2)$. Tedy počet všech nenasycených převedení je $\mathcal{O}(n^3)$. □

Tento odhad je hezký, ale stále není těsný a algoritmus se chová lépe. Dokažme si ještě jeden těsnější odhad na počet nenasycených převedení, užitečný zvláště v řídkých grafech.

Lemma N": Počet nenasycených převedení je $\mathcal{O}(n^2\sqrt{m})$.

Důkaz: Zaveďme fáze stejně jako v důkazu předchozí verze lemmatu a rozdělme je na dva druhy: laciné a drahé podle toho, kolik se v nich provede nenasycených převedení. Pro každý druh fází přitom odhadneme celkový počet převedení jiným způsobem.

Nechť k je nějaké kladné číslo, jehož hodnotu určíme později. *Laciné fáze* budou ty, během nichž se provede nejvýše k nenasycených převedení. *Drahé fáze* budou ty ostatní, tedy takové, ve kterých se provede více jak k nenasycených převedení.

Teď potřebujeme odhadnout, kolik nás budou stát oba typy fází. Začněme těmi jednoduššími – lacinými. Jejich počet shora odhadneme počtem všech fází, tedy $\mathcal{O}(n^2)$. Nenasycených převedení se během jedné laciné fáze provede nejvíce k. Během všech laciných fází dohromady jich proto bude $\mathcal{O}(n^2k)$.

Pro počet nenasycených převedení v drahých fázích si zaveďme nový potenciál:

$$\Psi := \sum_{\substack{v \neq z,s \\ f^\Delta(v) \neq 0}} p(v),$$

kde $p(v)$ je počet vrcholů u, které nejsou výše než v. Neboli:

$$p(v) = |\{u \in V \mid h(u) \leq h(v)\}|.$$

Tedy platí, že $p(v)$ je vždy nezáporné nikdy nepřesáhne počet všech vrcholů n. Proto Ψ bude také vždy nezáporné a nepřekročí n^2. Rozmysleme si, jak bude potenciál ovlivňován operacemi algoritmu:

- *Inicializace:* Počáteční potenciál je nejvýše n^2.
- *Zvednutí vrcholu* v: Hodnota $p(v)$ se zvýší nejvýše o n a všechna ostatní $p(w)$ se buďto nezmění, nebo klesnou o 1. Bez ohledu na přebytky vrcholů se tedy potenciál zvýší nejvýše o n.
- *Nasycené převedení* po hraně uv: Hodnoty $p(\ldots)$ se nezmění, ale mění se přebytky – vrcholu u se snižuje, vrcholu v zvyšuje. Z potenciálu proto může zmizet člen $p(u)$ a naopak přibýt $p(v)$. Potenciál Ψ tedy vzroste nejvýše o n.
- *Nenasycené převedení* po hraně uv: Hodnoty $p(\ldots)$ se opět nemění. Přebytek v u se vynuluje, což sníží Ψ o $p(u)$. Přebytek v se naopak zvýší, takže pokud byl předtím nulový, Ψ se zvýší o $p(v)$. Celkově tedy Ψ klesne alespoň o $p(u) - p(v)$.

  Teď využijeme toho, že pokud převádíme po hraně uv, má tato hrana spád 1. Výraz $p(u) - p(v)$ tedy udává počet vrcholů na hladině $h(u)$, což je nejvyšší hladina s přebytkem. Z předchozího důkazu víme, že těchto vrcholů je alespoň tolik, kolik je nenasycených převedení během dané fáze.

  Z toho plyne, že nenasycená převedení provedená během drahých fází sníží potenciál alespoň o k. Ty v laciných fázích ho nesnižují tak výrazně, ale určitě ho nezvýší.

Potenciál Ψ se tedy může zvětšit pouze při operacích inicializace, zvednutí a nasyceného převedení. Inicializace přispěje n^2. Všech zvednutí se provede celkem $\mathcal{O}(n^2)$

a každé zvýší potenciál nejvýše o n. Nasycených převedení se provede celkem $\mathcal{O}(nm)$ a každé zvýší potenciál taktéž nejvýše o n. Celkem se tedy Ψ zvýší nejvýše o

$$n^2 + n \cdot \mathcal{O}(n^2) + n \cdot \mathcal{O}(nm) = \mathcal{O}(n^3 + n^2 m).$$

Teď využijeme toho, že Ψ je nezáporný potenciál, tedy když ho každé nenasycené převedení v drahé fázi sníží Ψ alespoň o k, může takových převedení nastat nejvýše $\mathcal{O}(n^3/k + n^2 m/k)$. To nyní sečteme s odhadem pro laciné fáze a dostaneme, že všech nenasycených převedení proběhne

$$\mathcal{O}\left(n^2 k + \frac{n^3}{k} + \frac{n^2 m}{k}\right) = \mathcal{O}\left(n^2 k + \frac{n^2 m}{k}\right)$$

(využili jsme toho, že v grafech bez izolovaných vrcholů je $n = \mathcal{O}(m)$, a tedy $n^3 = \mathcal{O}(n^2 m)$).

Tento odhad ovšem platí pro libovolnou volbu k. Proto zvolíme takové k, aby byl co nejnižší. Jelikož první člen s rostoucím k roste a druhý klesá, asymptotické minimum nastane tam, kde se tyto členy vyrovnají, tedy když $n^2 k = n^2 m/k$.

Nastavíme tedy $k = \sqrt{m}$ a získáme kýžený odhad $\mathcal{O}(n^2\sqrt{m})$. □

Cvičení

1. Navrhněte implementaci vylepšeného Goldbergova algoritmu se zvedáním nejvyššího vrcholu s přebytkem. Snažte se dosáhnout časové složitosti $\mathcal{O}(n^2\sqrt{m})$.