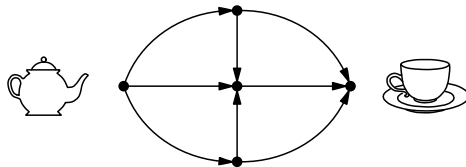


1. Toky v sítích

Posluchači jisté univerzity měli rádi čaj. Tak si řekli, že by do každé přednáškové místnosti mohli zavést čajovod. Nemuselo by to být komplikované: ve sklepech velikánská čajová konvice, všude po budově trubky. Tlustší by vedly od konvice do jednotlivých pater, pak by pokračovaly tenčí do jednotlivých poslucháren. 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. Hraný představují jednotlivé trubky s určenou kapacitou, ve vrcholech se trubky setkávají a větví. Máme na výběr, kolik tekutiny pošleme kterou trubkou, a 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 linka přeneše 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 spojitá můžeme považovat.

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

1.1. Definice toku

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

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

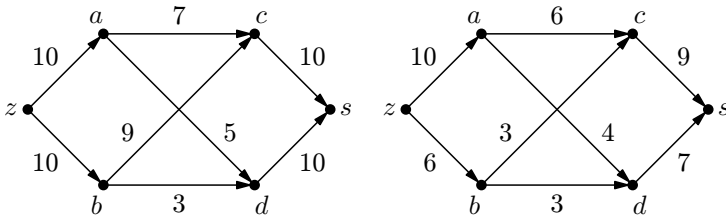
Podobně jako v předchozích kapitolách budeme počet vrcholů grafu značit n a počet hran m .

Mimo to budeme často předpokládat, že graf neobsahuje izolované vrcholy a je symetrický: je-li uv hranou grafu, je jí i vu . 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 $f : E \rightarrow \mathbb{R}_0^+$, 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 odeče („sít 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 pro ně zavedeme šikovní značení:

Definice: Pro libovolnou funkci $f : E \rightarrow \mathbb{R}$ 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 $|f|$ a bude rovna přebytku spotřebiče $f^\Delta(s)$. Říká nám tedy, kolik tekutiny přiteče do spotřebiče a nevrátí se zpět do sítě.

Jelikož síť těsní, mělo by být jedno, zda velikost toku měříme u spotřebiče, nebo u zdroje. Vskutku – přebytky zdroje a spotřebiče se liší pouze znaménkem:

Lemma: $f^\Delta(z) = -f^\Delta(s)$.

Důkaz: Uvážíme součet přebytků všech vrcholů

$$S = \sum_v f^\Delta(v).$$

Podle Kirchhoffova zákona může přebytek nenulový pouze ve zdroji a spotřebiči, takže tato suma musí být rovna $f^\Delta(z) + f^\Delta(s)$. Současně ale musí být nulová: je to

totiž součet nějakých kladných a záporných toků po hranách, přičemž každá hrana přispěje jednou kladně (ve vrcholu, do kterého vede) a jednou záporně (ve vrcholu, odkud vede). \square

Poznámka: Když vyslovíme nějakou definici, měli bychom se ujistit, že definovaný objekt existuje. S tokem jako takovým je to snadné: definici toku splňuje v libovolné síti všude nulová funkce. Maximální tok je ale ošidnější: i v jednoduché síti můžeme najít nekonečně mnoho různých toků. Není tedy a priori jasné, že nějaký z nich musí být největší. Zde by pomohla matematická analýza (cvičení 2), ale my na to raději půjdeme 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 „čajový“ příklad z úvodu kapitoly: v něm totiž bylo více spotřebičů. Ukažte, jak takový příklad pomocí našeho modelu toků popsat.
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} , pročež musí nabývat minima i maxima.

1.2. Fordův-Fulkersonův algoritmus

Nejjednodušší z algoritmů na hledání maximálního toku 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. Zvolíme

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

a tok po každé hraně cesty zvýšíme o ε . Přesněji řečeno, definujeme nový tok f' takto:

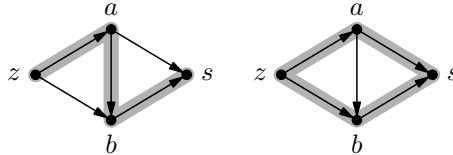
$$f'(e) := \begin{cases} f(e) + \varepsilon & \text{pro } e \in P \\ f(e) & \text{pro } e \notin P \end{cases}$$

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

Například v toku na obrázku 1.2 můžeme využít cestu $zbc s$ 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 skutečně 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 ukazují pravý obrázek, maximální tok má velikost 2.



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

Tuto překerní situaci by zachránilo, kdybychom mohli poslat tok velikosti 1 proti směru hrany ab . Pak bychom tok z levého obrázku zlepšili po cestě $zbas$ a získali bychom maximální tok z pravého obrázku. Posílat proti směru hrany ve skutečnosti nemůžeme, ale stejný efekt bude mít odečtení jedničky od toku po směru hrany.

Rozšíříme tedy náš algoritmus, aby byl ochoten nejen přičítat po směru hran, ale také odčítat proti směru. Pokud chceme zvýšit tok z u do v , můžeme přičíst k $f(uv)$ nejvýše $c(uv) - f(uv)$, abychom nepřekročili kapacitu. Podobně odečíst od $f(vu)$ smíme nejvýše $f(vu)$, abychom nevytvořili záporný tok. Součtu těchto dvou možných zlepšení se říká rezerva:

Definice: *Rezerva hrany uv* je číslo $r(uv) := c(uv) - f(uv) + f(vu)$. Hraně s nulovou rezervou budeme říkat *nasyčená*, hraně s kladnou rezervou *nenasyčená*. O cestě řekneme, že je *nasyčená*, pokud je nasyčená alespoň jedna její hrana; jinak mají všechny hrany kladné rezervy a cesta je *nenasyčená*.

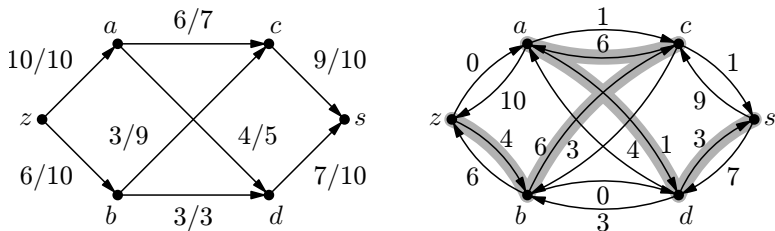
Roli zlepšujících cest tedy budou hrát nenasyčené cesty. Budeme je opakovaně hledat a tok po nich zlepšovat. Tím dostaneme algoritmus, který objevili v roce 1954 Lester Ford a Delbert Fulkerson.

Algoritmus FORDFULKERSON

Vstup: Síť (V, E, c, z, s)

1. $f \leftarrow$ libovolný tok, např. všude nulový
2. Dokud existuje nenasyčená cesta P ze z do s , opakujeme:
3. $\varepsilon \leftarrow \min\{r(e) \mid e \in P\}$ \triangleleft spočítáme rezervu celé cesty
4. Pro všechny hrany $uv \in P$:
5. $\delta \leftarrow \min\{f(vu), \varepsilon\}$ \triangleleft kolik můžeme odečíst v protisměru
6. $f(vu) \leftarrow f(vu) - \delta$
7. $f(uv) \leftarrow f(uv) + \varepsilon - \delta$ \triangleleft zbytek přičteme po směru

Výstup: Maximální tok f



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

Rozbor algoritmu

Abychom dokázali, že algoritmus vydá maximální tok, nejprve si musíme ujasnit, že se vždy zastaví. Nemohlo by se třeba stát, že bude tok bude růst donekonečna o stále menší a menší hodnoty?

- Pakliže jsou všechny kapacity celá čísla, vypočtené toky budou také celočíselné. Velikost toku se proto v každém kroku zvětší alespoň o 1 a algoritmus se zastaví po nejvýše tolika krocích, kolik je nějaká horní mez pro velikost maximálního toku – např. součet kapacit všech hran vedoucích do stoku. (O moc rychlejší ale být nemusí, jak uvidíme ve cvičení 1.)
- Pro racionální kapacity využijeme jednoduchý trik. Nechtě 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 (cvičení 2).

Pro celočíselné a racionální kapacity se tedy algoritmus zastaví a vydá jako výsledek nějaký tok f . Abychom dokázali, že tento tok je maximální, povoláme na pomoc řezy. Definujeme je podobně jako v kapitole o minimálních kostrách, ale tentokrát pro orientované grafy se zdrojem a stokem.

Definice: Pro libovolné dvě množiny vrcholů A a B označíme $E(A, B)$ 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 (přesněji řečeno *elementární řez*, viz cvičení 5) je množina hran, kterou lze zapsat jako $E(A, B)$ pro nějaké dvě množiny vrcholů A a B . Tyto množiny musí

být disjunktní, musí dohromady obsahovat všechny vrcholy a navíc v A musí ležet zdroj a v B 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 $E(A, B)$ a každý tok f platí $f^\Delta(A, B) = |f|$.

Důkaz: Opět šikovně sečteme přebytky 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 $v \in B$ do jiného vrcholu $w \in B$ k sumě přispěje jednou kladně a jednou záporně; hrany ležící zcela 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 a zdroj v B neleží. \square

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

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

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

Důsledek: Pokud $|f| = c(A, B)$, pak je tok f maximální a řez $E(A, B)$ minimální (s nejmenší možnou kapacitou). Velikost toku f totiž nelze zvětšit nad kapacitu řezu $E(A, B)$, zatímco řez $E(A, B)$ nejde zmenšit pod velikost toku f .

Takže 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 takovou dvojici toku s řezem vždy najdeme.

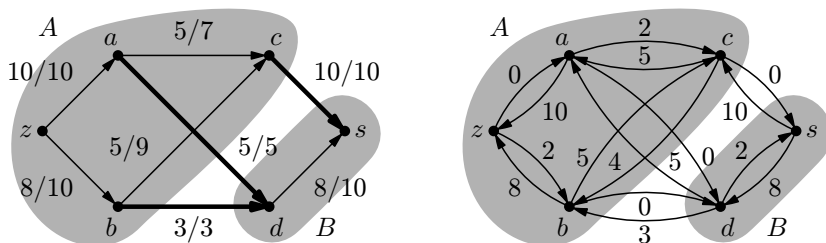
Lemma: Pokud se Fordův-Fulkersonův algoritmus zastaví, vydá maximální tok.

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

Všimneme si, že množina $E(A, B)$ je řez: Zdroj z leží v A , protože ze z do z existuje cesta nulové délky, která je tím pádem nenasyčená. Spotřebič musí ležet v B , neboť jinak by existovala nenasyčená 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 nenasyčené cesty ze zdroje do u s touto hranou by vznikla nenasyčená cesta ze zdroje do v , takže vrchol v by také musel ležet v A , a nikoliv v B .

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



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

Poznámka: Kdyby přišel kouzelník a rovnou ze svého klobouku vytáhl maximální tok, jen těžko by skeptické publikum přesvědčoval o jeho maximalitě. Fordův-Fulkersonův algoritmus to má snazší: k toku vydá i *certifikát* jeho maximality, totiž příslušný minimální řez. To, že tok i řez jsou korektní a že jejich velikosti se rovnají, může publikum ověřit v lineárním čase.

Nyní konečně 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. \square

To, že umíme najít celočíselné řešení, není vůbec samozřejmé. U mnoha problémů je racionální varianta snadná, zatímco celočíselná velmi obtížná (viz třeba celočíselné lineární rovnice v kapitole ??). Teď si ale chvíli užijeme, 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. Lze dokonce zařídit, aby k maximálnímu toku ani nekonvergoval.
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 dokonce vrcholově disjunktní (až na krajní vrcholy).
5. Obecná definice řezu říká, že řez je množina hran grafu, po jejímž odebrání se graf rozpadne na více komponent (případně máme-li určený zdroj a stok,

skončí oba v různých komponentách). Srovnejte tuto definici s naší definicí elementárního řezu. Ukažte, že existují i neelementární řezy. Také ukažte, že jsou-li kapacity všech hran kladné, pak každý minimální řez je elementární.

6. Profesor Forderson si přečetl začátek tohoto oddílu a pokusil se chybný algoritmus (který zlepšuje tok pouze po směru hran) zachránit tím, že bude vybírat *nejkratší* zlepšující cesty. Ukažte, že zlepšovák pana profesora nefunguje!
7. Pokračujme ve vynálezech profesora Fordersona z předchozího cvičení. Existuje vůbec nějaká posloupnost zlepšujících cest po směru hran, která vede k maximálnímu toku? Pokud ano, platí to i pro posloupnost nejkratších zlepšujících cest?
- 8.* 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.)
- 9.*^{*} Přímočará implementace Fordova-Fulkersonova algoritmu bude nejspíš graf prohledávat do šířky, takže vždy najde *nejkratší* nenasycenou cestu. Pak překvapivě platí, že algoritmus zlepší tok jen $\mathcal{O}(nm)$ -krát, než se zastaví. Návod k důkazu: Necht $\ell(u)$ 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.

1.3. Největší párování v bipartitních grafech

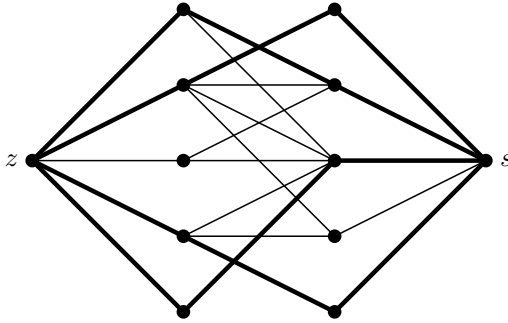
Problém maximálního toku je zajímavý nejen sám o sobě, ale také tím, že na něj můžeme elegantně převádět jiné problémy. Jeden takový si ukážeme a rovnou při tom využijeme celočíselnost.

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

Párování v bipartitních grafech má zjevné aplikace: Jednu partitu může tvořit třeba množina mlounů, druhou množina zákusků a hrany říkají, kdo má na co chuť. Párování pak odpovídá tomu, že mlounům rozdáme jejich oblíbené zákusky tak, aby žádný mloun nedostal dva a žádný zákusek nebyl sněden vícekrát. Přirozeně chceme nalézt párování o co nejvíce hranách.

Mějme tedy nějaký bipartitní graf (V, E) . Přetvoříme ho na síť (V', E', z, s, c) následovně:

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

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 $\mathcal{O}(n'm') = \mathcal{O}(nm)$. \square

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 rozestavení 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. Kostky je povoleno otáčet.
- 5.* 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?
- 6.* 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 nalézt maximální tok Fordovým-Fulkersonovým algoritmem. Začali jsme s tokem nulovým a postupně jsme ho zvětšovali. Pokaždé jsme v síti našli *nenasyčenou cestu*, tedy takovou, na níž mají všechny hrany kladnou rezervu. Podél cesty jsme pak tok zlepšili.

Nepríjemné je, že může trvat velice dlouho, než se tímto způsobem dobereme k maximálnímu toku. Pro obecné reálné kapacity se to dokonce nemusí stát vůbec. Proto odvodíme 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 nějakou 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 wv . To vede k následujícímu popisu toků.

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

Pozorování: Průtoky mají následující 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): Necht' funkce $f^* : E \rightarrow \mathbb{R}$ 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 využijeme (1) a u prohodíme s v . 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. \square

Důsledek: Místo toků tedy stačí uvažovat průtoky hranami. Tím se ledacos formálně zjednoduší: 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: *Sít' 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 počítat nemůžeme, ale průtoky po jednotlivých hranách už ano. Pro každou hranu e položíme $h^*(e) := f^*(e) + g^*(e)$. Nahlédneme, 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 $h^*(uv) = f^*(uv) + g^*(uv) \leq c(uv)$.
- (4) 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. \square

Poznámka: Zlepšení po nenasyčené cestě je speciálním případem tohoto postupu – odpovídá toku v síti rezerv, který je konstantní na jedné cestě a všude jinde nulový.

Dinicův algoritmus

Dinicův algoritmus začne s nulovým tokem a bude ho vylepšovat pomocí nějakých pomocných toků v síti rezerv, až se dostane k maximálnímu toku. Počet potřebných iterací přitom bude záviset na tom, jak „vydatné“ 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. Můžeme si představit, že provádíme na jedinou mnoho iterací Fordova-Fulkersonova algoritmu pro všechny nejkratší cesty.

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:

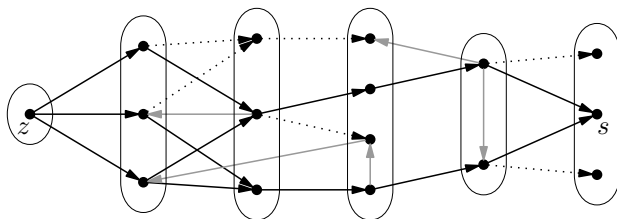
Algoritmus DINIC

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 žádná taková cesta neexistuje, zastavíme se a vrátíme výsledek f .
 6. Pročistíme síť R .
 7. $g \leftarrow$ blokuující 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 podle vzdálenosti od zdroje. 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 hrany během čištění zmizí, plné zůstanou.

Procedura ČIŠTĚNÍ SÍTĚ

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. $F \leftarrow \{v \neq z, s \mid \deg^{\text{out}}(v) = 0\}$ \triangleleft 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^{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.

Procedura BLOKUJÍCÍ TOK

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. $\varepsilon \leftarrow \min_{e \in P} (r(e) - g(e))$
4. Pro všechny $e \in P$: $g(e) \leftarrow g(e) + \varepsilon$.
5. Pokud pro kteroukoliv e nastalo $g(e) = r(e)$, smažeme e z R .
6. 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. \square

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 každé ze sítě vypadne alespoň jedna hrana (ta, na níž se v kroku 3 nabývalo minimum) a už se tam

nevrátí. Jelikož síť je vrstevnatá, nalézt jednu cestu stihneme v $\mathcal{O}(n)$. Celkem tedy spotřebujeme čas $\mathcal{O}(nm)$ plus čištění, které jsme ale už započítali.

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

Zbývá 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 vzroste po každé fázi 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. Necht' 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 a 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 dostatečně 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 v R_{i+1} objevit, vedou o jednu vrstvu zpět. Jenže každá cesta ze zdroje do spotřebiče, která se alespoň jednou vrátí o vrstvu zpět, musí mít délku alespoň $\ell + 2$ (spotřebič je v ℓ -té vrstvě a neexistují hrany, které by vedly o více než 1 vrstvu dopředu). \square

Důsledek: Proběhne maximálně n fází.

Důkaz: Cesta ze z do s obsahuje nejvýše n hran, takže k prodloužení cesty dojde nejvýše n -krát. \square

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

Důkaz: Podle právě vysloveného důsledku proběhne nejvýše n fází. Každá z nich 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. \square

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). Uvedme 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. 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.

2. Odsimulujte běh Dinicova algoritmu na svém řešení cvičení 1.2.1.
3. Dokažte, že pro jednotkové kapacity Dinicův algoritmus doběhne v čase $\mathcal{O}(nm)$.
4. Dokažte totéž pro celočíselné kapacity omezené konstantou.
5. Blokující tok lze také sestrojít pomocí prohledávání do hloubky. Pokaždé, když projdeme hranou, přepočítáme průběžné minimum. Pokud najdeme stok, vrátíme 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 tak, aby zůstala zachovaná časová složitost $\mathcal{O}(n^2m)$.
6. Sestrojte vrstevnatou síť, v níž hledání blokujícího toku trvá $\Omega(nm)$.
7. Sestrojte síť, na níž Dinicův algoritmus provede $\Omega(n)$ fází.
8. Zkombinujte předchozí dvě cvičení a vytvořte síť, na níž Dinicův algoritmus běží v čase $\Omega(n^2m)$.
- 9:** *Algoritmus tří Indů*: Blokující tok ve vrstevnaté síti lze nalézt chytřejším způsobem v čase $\mathcal{O}(n^2)$, čímž zrychlíme celý Dinicův algoritmus na $\mathcal{O}(n^3)$. Následuje stručný popis, doplňte k němu detaily.

Pro každý vrchol v definujeme $r^+(v)$ jako součet rezerv na všech hranách vedoucích do v . Nechť dále $r^-(v)$ je totéž přes hrany vedoucí z v a $r(v) = \min(r^+(v), r^-(v))$ „rezerva vrcholu“. Pokud je $r(v)$ všude 0, tok už je blokující.

V opačném případě opakovaně vybíráme nejmenší $r(v)$ a snažíme se ho vynulovat. Potřebujeme tedy dopravit $r(v)$ jednotek toku ze zdroje do v a totéž množství z v do stoku. Popišme dopravu do stoku (ze zdroje postupujeme symetricky): ve vrcholech udržujeme plán $p(w)$, který říká, kolik potřebujeme z w dopravit do stoku. Na začátku je $p(v) = r(v)$ a všechna ostatní $r(w) = 0$. Procházíme po vrstvách od v ke stoku a pokaždé plán převedeme po hranách s kladnou rezervou do vrcholů v další vrstvě. Jelikož $r(v) \leq r(w)$ pro všechna w , vždy nám to vyjde. Průběžně čistíme slepé uličky.

1.5. Goldbergův algoritmus

Představíme si ještě jeden algoritmus pro hledání maximálního toku v síti, navržený Andrewem Goldbergem. 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 rozbořením 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 $f : E \rightarrow \mathbb{R}_0^+$ 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. V průběhu výpočtu se tedy potřebujeme postupně zbavit nenulových přebytků ve všech vrcholech kromě zdroje a spotřebiče. K tomu bude sloužit následující operace:

Definice: *Převedení přebytku* po hraně uv jsme ochotni provést, pokud $f^\Delta(u) > 0$ a $r(uv) > 0$. Proběhne 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 přepravili do spotřebiče, nebo je naopak 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 nenасыcená 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 (V, E, c, z, s)

1. Nastavíme počáteční výšky: \triangleleft 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: \triangleleft všechny hrany ze z na maximum
5. $f \leftarrow$ všude nulová funkce
6. $f(zv) \leftarrow c(zv)$, kdykoliv $zv \in E$
7. Dokud existuje vrchol $u \neq z, s$ takový, že $f^\Delta(u) > 0$:
8. Pokud existuje hrana uv s $r(uv) > 0$ 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 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 neporušuje 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. \square

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 nebo do kopce. 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. \square

Lemma K (o korektnosti): Když se algoritmus zastaví, f je 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 stok 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. \square

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

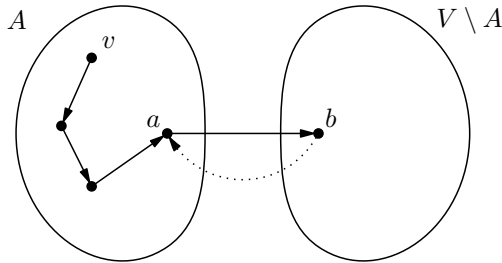
$$A := \{u \in V \mid \text{existuje nenasycená cesta z } v \text{ do } u\}.$$

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. Stačí tedy započítat pouze hrany vedoucí ven z A , nebo naopak zvenku dovnitř. Získáme:

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

Ukážeme, že první svorka je rovna nule (sledujme obrázek 1.8). 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.



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

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

Proto součet přebytků přes množinu A je menší nebo roven nule. Zároveň však v A leží aspoň jeden vrchol s kladným přebytkem, totiž v , tudíž v A musí být také nějaký vrchol se záporným přebytkem – a jediný takový je zdroj. Tím je dokázáno, že z leží v A , tedy že vede nenasycená cesta z vrcholu v do zdroje. \square

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 $2n$. Vrchol přitom zvedáme jen tehdy, má-li kladný

přebytek. Dle invariantu **C** musela v tomto okamžiku existovat nenasyčená cesta z v do zdroje. Ta nicméně překonávala spád alespoň n , ale mohla mít nejvýše $n - 1$ hran. Tudíž musela obsahovat nenasyčenou hranu se spádem alespoň 2 a máme spor s invariantem **S**. \square

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

Důkaz: Z předchozího invariantu plyne, že každý z n vrcholů mohl být zvednut nejvýše $2n$ -krát. \square

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 *nasyčené*, pokud po převodu rezerva $r(uv)$ klesla na nulu. V opačném případě je *nenasyčené*, a tehdy určitě klesne přebytek $f^\Delta(u)$ na nulu (to se nicméně může stát i při nasyčeném převedení).

Lemma S (nasyčená převedení): Nastane nejvýše nm nasyčených převedení.

Důkaz: Zvolíme hranu uv a spočítáme, kolikrát jsme po ní mohli nasyčeně převést.

Po prvním nasyčeném převedení z u do v se vynulovala rezerva hrany uv . V tomto okamžiku muselo být u výše než v , a dokonce víme, že bylo výše přesně o 1 (invariant **S**). Než nastane další převedení po této hraně, hrana musí opět získat nenulovou rezervu. Jediný způsob, jak k tomu může dojít, je převedením části přebytku z v zpátky do u . Na to se musí v dostat (alespoň o 1) výše než u . A abychom provedli nasyčené převedení znovu ve směru z u do v , musíme 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 nasyčenými převedeními po hraně uv musel být vrchol u alespoň dvakrát zvednut. Podle lemmatu **V** k tomu ale mohlo dojít nejvýše n -krát za celý výpočet, takže všech nasyčených převedení po hraně uv je nejvýše n a po všech hranách dohromady nejvýše nm . \square

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

Použijeme potenciálovou metodu, která se již osvědčila při amortizované analýze v oddílu ???. Pořídíme si *potenciál*, což bude 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.

Někdy bývá potenciál přímočará funkce, ale v následujícím lemmatu bude trochu složitější. Zvolíme ho tak, aby operace, jejichž počty už známe (zvednutí, nasyčené převedení), přispívaly nanejvýš malými kladnými čísly, a nenasyčená převedení potenciál vždy snižovala.

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

Důkaz: Uvažujme následující potenciál:

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

Každý vrchol s kladným přebytkem tedy přispívá svou výškou. Sledujme, jak se tento potenciál během výpočtu 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, musí mít kladný přebytek, takže vrchol do sumy již přispíval. Teď jen přispěje číslem o 1 vyšším.) Již víme, že za celý průběh algoritmu nastane maximálně $2n^2$ zvednutí, protože zvedáním vrcholů zvýšíme potenciál dohromady nejvýše o $2n^2$.
- Nasycené převedení zvýší Φ nejvýše o $2n$: Pokud vrchol v měl původně nulový přebytek, bude mít teď kladný a do sumy začne přispívat; tím se suma zvýší o $h(v) \leq 2n$. Vrchol u již přispíval a buďto bude přispívat nadále, nebo se suma sníží. Podle lemmatu **S** nastane nejvýše nm nasycených převedení a ta celkově potenciál zvýší maximálně o $2n^2m$.
- Konečně když převádíme po hraně uv nenasyčeně, tak od potenciálu určitě odečteme výšku vrcholu u (neboť se vynuluje přebytek ve vrcholu u) a možná přičteme výšku vrcholu v (nevíme, zda tento vrchol předtím měl přebytek). Jenže $h(v) = h(u) - 1$, a proto nenasyčené převedení potenciál vždy sníží alespoň o jedna.

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

Implementace

Zbývá vyřešit, jak síť a výšky reprezentovat, abychom dokázali rychle hledat vrcholy s přebytkem a nenasyčené 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 budeme udržovat seznam $L(u)$. Ten bude uchovávat všechny nenasyčené 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 může způsobit, že nějaká hrana s kladnou rezervou, která původně vedla po rovině, začne vést z u dolů. Nebo se naopak může stát, že hrana, která původně vedla s kopce do u , najednou vede po rovině. Musíme proto 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 zvednutí je sice drahé, ale zase je jich 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 $\mathcal{O}(n^3 + nm + n^2m) = \mathcal{O}(n^2m)$. Jakmile se algoritmus zastaví, podle lemmatu **K** vydá maximální tok. \square

Cvičení

1. Rozberte 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 drobnou úpravou lze Goldbergův algoritmus ještě zrychlit. Postačí ze všech vrcholů s přebytkem pokaždé vybírat ten nejvyšší.

Při rozboru časové složitosti původního algoritmu hrál nejvýznamnější roli člen $\mathcal{O}(n^2m)$ za nenasycená převedení. Ukážeme, že ve vylepšeném algoritmu jich nastane řádově méně.

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*:

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

Rozdělíme běh algoritmu na *fáze*. Každá fáze končí tím, že se H změní. Buďto se H zvýší, což znamená, že nějaký vrchol s přebytkem v nejvyšší hladině byl o 1 zvednut, anebo 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 $\mathcal{O}(n^3)$. \square

Ve skutečnosti je i tento odhad trochu nadhodnocený. Trochu složitějším argumentem lze dokázat těsnější odhad, který se hodí zvláště u řídkých grafů.

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

Důkaz: Zavedeme fáze stejně jako v důkazu předchozí verze lemmatu a rozdělíme je na dva druhy. Pro každý druh pak odhadneme celkový počet převedení jiným způsobem.

Nechť k je nějaké kladné číslo, jehož hodnotu určíme později. *Laciné* nazveme ty fáze, během nichž se provede nejvýše k nenasycených převedení. *Drahé* fáze budou všechny ostatní.

Nejprve rozebereme chování laciných fází. Jejich počet shora odhadneme počtem všech fází, tedy $\mathcal{O}(n^2)$. Nenasycených převedení se během jedné laciné fáze provede nejvíce k , za všechny laciné fáze dohromady to činí $\mathcal{O}(n^2k)$.

Pro počet nenasycených převedení v drahých fázích zavedeme 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 . Jelikož $p(v)$ je nezáporné a nikdy nepřesáhne počet všech vrcholů, potenciál Ψ bude také vždy nezáporný 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 $p(\dots)$ 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 $p(\dots)$ 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 $p(u) - p(v)$.

Tedy 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 nenasycené převedení provedené během drahé fáze sníží potenciál alespoň o k . Převedení v laciných fázích ho nesnižuje tak výrazně, ale důležité je, že ho určitě nezvýší.

Potenciál Ψ se tedy může zvětšit pouze při operacích inicializace, zvednutí a nasyčeného převedení. Inicializace přispěje n^2 . Všechna zvednutí se provede celkem $\mathcal{O}(n^2)$ a každé zvýší potenciál nejvýše o n . Nasyčených převedení se provede celkem $\mathcal{O}(nm)$ a každé zvýší potenciál taktéž nejvýše o n . Celkem se tedy Ψ zvýší nejvýše o:

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

Tedy 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 $\mathcal{O}(n^3/k + n^2m/k)$. To nyní sečteme s odhadem pro laciné fáze a dostaneme, že všech nenasycených převedení proběhne

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

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

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 $n^2k = n^2m/k$.

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 $\mathcal{O}(n^2\sqrt{m})$.

1.7. Další cvičení

1. *Svišti*: Na louce je n svišťů a m děr v zemi (obojí je zadáno jako body v rovině nebo raději body v nepřilíh velké celočíselné mřížce). Když se objeví orel, zvládně svišť uběhnout pouze d metrů, než bude uloven. Kolik maximálně svišťů se může

zachránit útekem do díry, když jedna díra pojme nejvýše jednoho sviště? A co když pojme k svišťů?

2. *Parlamentní kluby:* V parlamentu s n poslanci je m různých klubů. Jeden poslanec může být členem mnoha různých klubů. Každý klub nyní potřebuje zvolit svého předsedu a tajemníka tak, aby všichni předsedové a tajemníci byli navzájem různé osoby (tedy aby nikdo „neseděl na více křeslech“). Navrhněte algoritmus, který zvolí všechny předsedy a tajemníky, případně oznámí, že řešení neexistuje. Mimochodem, za jakých podmínek je existence řešení garantována?
3. *Dopravní problém:* Uvažujme továrny T_1, \dots, T_p a obchody O_1, \dots, O_q . Všichni vyrábějí a prodávají 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.
- 4.* *Průchod šachovnicí:* Je dána šachovnice $n \times n$, kde některá políčka jsou nepřístupná. Celý dolní řádek je obsazen figurkami, které se mohou hýbat o jedno pole dopředu, šikmo vlevo dopředu, či šikmo vpravo dopředu. V jednom tahu se všechny figurky naráz pohnou (mohou i zůstat stát na místě), na jednom políčku se však musí vyskytovat nejvýše jedna figurka. Ocitne-li se figurka na některém políčku horního řádku šachovnice, zmizí. Navrhněte algoritmus, který najde minimální počet tahů takový, že z šachovnice dokážeme odstranit všechny figurky, případně oznámí, že řešení neexistuje.
- 5.* *Minimální izolace:* Je dán nepříliš velký celočíselný kvádr (dejme tomu s objemem nejvýše 20 000) a v něm k nebezpečných jednotkových kostiček. Navrhněte algoritmus, který najde podmnožinu M kostiček kvádrů takovou, že každá nebezpečná kostička leží v M a zároveň M má minimální možný povrch (povrch měříme jako počet takových stěn kostiček z M , které nesousedí s jinou kostičkou z M).
- 6.* *Doly a továrny:* Uvažujeme o vybudování dolů D_1, \dots, D_p a továren T_1, \dots, 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 a pokud jsou v provozu všechny doly produkující tyto suroviny, vyděláme na továrně zisk t_j . Vymyslete algoritmus, jenž pro zadané ceny dolů, zisky továren a bipartitní graf závislostí továren na surovinách stanoví, které doly postavit, abychom vydělali co nejvíce.
- 7.* *Permanent matice* $n \times n$ je definovaný podobně jako determinant, jen bez alternance znamének. Nahlédněte, že na permanent se dá dívat jako na součet přes všechna rozestavení n neohrožujících se věží na políčka matice, přičemž sčítáme součiny políček pod věžemi. 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.