2. Dinicův algoritmus a jeho varianty

Maximální tok v síti už umíme najít pomocí Fordova-Fulkersonova algoritmu z minulé kapitoly. Nyní pojednáme o efektivnějším Dinicově algoritmu a o různých jeho variantách pro sítě ve speciálním tvaru.

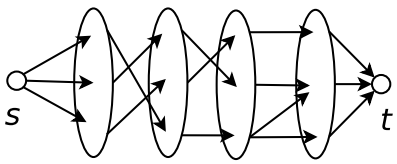
Dinicův algoritmus

Dinicův algoritmus je založen na následující myšlence: Ve Fordově-Fulkersonově algoritmu nemusíme přičítat jen zlepšující cesty, ale je možné přičítat rovnou zlepšující toky. Nejlépe toky takové, aby je nebylo obtížné najít, a přitom aby původní tok dostatečně obohatily. Vhodnými objekty k tomuto účelu jsou blokující toky:

Definice: *Blokující tok* je tok takový, že každá orientovaná st-cesta obsahuje alespoň jednu nasycenou hranu. [Tj. takový tok, který by našel F-F algoritmus, kdyby neuvažoval rezervy v protisměru.]

Algoritmus (Dinic):

1. Začneme s libovolným tokem f, například prázdným (všude nulovým).
2. Iterativně vylepšujeme tok pomocí zlepšujících toků: *(vnější cyklus)*
3. Sestrojíme síť rezerv: vrcholy a hrany jsou tytéž, kapacity jsou určeny rezervami v původní síti. Dále budeme pracovat s ní.
4. Najdeme nejkratší st-cestu. Když žádná neexistuje, skončíme.
5. Pročistíme síť, tj. ponecháme v ní pouze vrcholy a hrany na nejkratších st-cestách.
6. Najdeme v pročištěné síti blokující tok f_B:
7. $f_B \leftarrow$ *prázdný tok*
8. Postupně přidáváme st-cesty: *(vnitřní cyklus)*
9. Najdeme st-cestu. Např. hladově – „rovnou za nosem“.
10. Pošleme co nejvíce po nalezené cestě.
11. Smažeme nasycené hrany. (Pozor, smazáním hran mohou vzniknout slepé uličky, čímž se znečistí síť a nebude fungovat hladové hledání cest.)
12. Dočistíme síť.
13. Zlepšíme f podle f_B



Pročištěná síť rozdělená do vrstev

Složitost algoritmu: Označme l délku nejkratší st-cesty, n počet vrcholů sítě a m počet hran sítě.

- Jeden průchod vnitřním cyklem trvá $\mathcal{O}(l)$, což můžeme odhadnout jako $\mathcal{O}(n)$, protože vždy platí $l \leq n$.
- Vnitřní cyklus se provede maximálně m-krát, protože se vždy alespoň jedna hrana nasytí a ze sítě vypadne, takže krok 6 mimo podkroku 12 bude trvat $\mathcal{O}(mn)$.
- Čištění a dočišťování sítě dohromady provedeme takto:
  - Rozvrstvíme vrcholy podle vzdálenosti od s.
  - Zařízneme dlouhé cesty (delší, než do vrstvy obsahující t).
  - Držíme si frontu vrcholů, které mají $\deg^+ = 0$ či $\deg^- = 0$.
  - Vrcholy z fronty vybíráme a zahazujeme včetně hran, které vedou do/z nich. A případně přidáváme do fronty vrcholy, kterým při tom klesl jeden ze stupňů na 0. Vymažou se tím slepé uličky, které by vadily v podkroku 9.

  Takto kroky 5 a 12 dohromady spotřebují čas $\mathcal{O}(m)$.
- Jeden průchod vnějším cyklem tedy trvá $\mathcal{O}(mn)$.
- Jak za chvíli dokážeme, každým průchodem vnějším cyklem l vzroste, takže průchodů bude maximálně n a celý algoritmus tak poběží v čase $\mathcal{O}(n^2 m)$.

Korektnost algoritmu: Když se Dinicův algoritmus zastaví, nemůže už existovat žádná zlepšující cesta (viz krok 4) a tehdy, jak už víme z analýzy F-F algoritmu, je nalezený tok maximální.

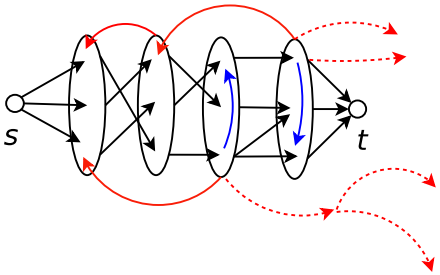
Věta: V každém průchodu Dinicova algoritmu vzroste l alespoň o 1.

Důkaz: Podíváme se na průběh jednoho průchodu vnějším cyklem. Délku aktuálně nejkratší st-cesty označme l. Všechny původní cesty délky l se během průchodu zaručeně nasytí, protože tok f_B je blokující. Musíme však dokázat, že nemohou vzniknout žádné nové cesty délky l nebo menší. V síti rezerv totiž mohou hrany nejen ubývat, ale i přibývat: pokud pošleme tok po hraně, po které ještě nic neteklo, tak v protisměru z dosud nulové rezervy vyrobíme nenulovou. Rozmysleme si tedy, jaké hrany mohou přibývat:

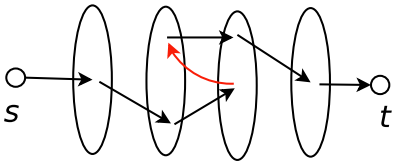
Vnější cyklus začíná s nepročištěnou sítí. Příklad takové sítě je na následujícím obrázku. Po pročištění zůstanou v síti jen černé hrany, tedy hrany vedoucí z i-té vrstvy do $(i+1)$-ní. Červené a modré[1] se zahodí.

Nové hrany mohou vznikat výhradně jako opačné k černým hranám (hrany ostatních barev padly za oběť pročištění). Jsou to tedy vždy zpětné hrany vedoucí z i-té vrstvy do $(i-1)$-ní. Vznikem nových hran by proto mohly vzniknout nové st-cesty, které používají zpětné hrany. Jenže st-cesta, která použije zpětnou hranu, musí alespoň jednou skočit o vrstvu zpět a nikdy nemůže skočit o více než jednu vrstvu dopředu, a proto je její délka alespoň $l+2$. Tím je věta dokázána. ♡

[1] Modré jsou ty, které vedou v rámci jedné vrstvy, červené vedou zpět či za spotřebič či do slepých uliček. Při vytištění na papír vypadají všechny černě.



Nepročištěná síť. Obsahuje zpětné hrany, hrany uvnitř vrstvy a slepé uličky.



Cesta užívající novou zpětnou hranu

Poznámky

- Není potřeba tak puntičkářské čištění. Vrcholy se vstupním stupněm 0 nám nevadí – stejně se do nich nedostaneme. Vadí jen vrcholy s výstupním stupněm 0, kde by mohl havarovat postup v podkroku 9.
- Je možné dělat prohledávání a čištění současně. Jednoduše prohledáváním do hloubky: „Hrrr na ně!“ a když to nevyjde (dostaneme se do slepé uličky), kus ustoupíme a při ústupu čistíme síť odstraňováním slepé uličky.
- Už při prohledávání si rovnou udržujeme minimum z rezerv a při zpáteční cestě opravujeme kapacity. Snadno zkombinujeme s prohledáváním do hloubky.
- V průběhu výpočtu udržujeme jen síť rezerv a tok vypočteme až nakonec z rezerv a kapacit.
- Když budeme chtít hledat minimální řez, spustíme po Dinicovu algoritmu ještě jednu iterací F-F algoritmu.

Speciální sítě (ubíráme na obecnosti)

Při převodu různých úloh na hledání maximálního toku často dostaneme síť v nějakém speciálním tvaru – třeba s omezenými kapacitami či stupni vrcholů. Podíváme se proto podrobněji na chování Dinicova algoritmu v takových případech a ukážeme, že často pracuje překvapivě efektivně.

Jednotkové kapacity: Pokud síť neobsahuje cykly délky 2 (dvojice navzájem opačných hran), všechny rezervy jsou jen 0 nebo 1. Pokud obsahuje, mohou rezervy být i dvojky, a proto síť upravíme tak, že ke každé hraně přidáme hranu opačnou s nulovou kapacitou a rezervu proti směru toku přiřkneme jí. Vzniknou tím sice paralelní

hrany, ale to tokovým algoritmům nikterak nevadí.[2]

Při hledání blokujícího toku tedy budou mít všechny hrany na nalezené st-cestě stejnou, totiž jednotkovou, rezervu, takže vždy z grafu odstraníme celou cestu. Když máme m hran, počet zlepšení po cestách délky l bude maximálně m/l. Proto složitost podkroků 9, 10 a 11 bude $m/l \cdot \mathcal{O}(l) = \mathcal{O}(m)$.[3] Tedy pro jednotkové kapacity dostáváme složitost $\mathcal{O}(nm)$.

Jednotkové kapacity znovu a lépe: Vnitřní cyklus lépe udělat nepůjde. Je potřeba alespoň lineární čas pro čištění. Můžeme se ale pokusit lépe odhadnout počet iterací vnějšího cyklu.

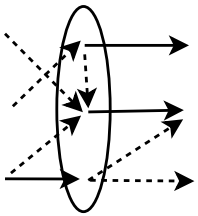
Sledujme stav sítě po k iteracích vnějšího cyklu a pokusme se odhadnout, kolik iterací ještě algoritmus udělá. Označme l délku nejkratší st-cesty. Víme, že $l > k$, protože v každé iteraci vzroste l alespoň o 1.

Máme tok f_k a chceme dostat maximální tok f. Rozdíl $f - f_k$ je tok v síti rezerv (tok v původní síti to ovšem být nemusí!), označme si ho f_R. Každá iterace velkého cyklu zlepší f_k alespoň o 1. Tedy nám zbývá ještě nejvýše $|f_R|$ iterací. Proto bychom chtěli omezit velikost toku f_R. Například řezem.

Najdeme v síti rezerv nějaký dost malý řez C. Kde ho vzít?[4] Počítejme jen hrany zleva doprava. Těch je jistě nejvýše m a tvoří alespoň k rozhraní mezi vrstvami. Tedy existuje rozhraní vrstev s nejvýše m/k hranami[5]. Toto rozhraní je řez. Tedy existuje řez C, pro nějž $|C| \leq m/k$, a algoritmu zbývá maximálně m/k dalších kroků. Celkový počet kroků je nejvýš $k + m/k$, takže stačí zvolit $k = \sqrt{m}$ a získáme odhad na počet kroků $\mathcal{O}(\sqrt{m})$.

Tím jsme dokázali, že celková složitost Dinicova algoritmu pro jednotkové kapacity je $\mathcal{O}(m^{3/2})$. Tím jsme si pomohli pro řídké grafy.

Jednotkové kapacity a jeden ze stupňů roven 1: Úlohu hledání maximálního párování v bipartitním grafu, případně hledání vrcholově disjunktních cest v obecném grafu lze převést (viz předchozí kapitola) na hledání maximálního toku v síti, v níž má každý vrchol $v \neq s, t$ buďto vstupní nebo výstupní stupeň roven jedné. Pro takovou síť můžeme předchozí odhad ještě trošku upravit. Pokusíme se nalézt v síti po k krocích nějaký malý řez. Místo rozhraní budeme hledat jednu malou vrstvu a z malé vrstvy vytvoříme malý řez tak, že pro každý vrchol z vrstvy vezmeme tu hranu, která je ve svém směru sama.



[2] Často se to implementuje tak, že protisměrné hrany vůbec nevytvoříme a když hranu nasytíme, tak v síti rezerv prostě obrátíme její orientaci.

[3] Nebo by šlo argumentovat, tím že každou hranu použijeme jen 1×.

[4] Přeci v řeznictví. Kdepak, spíše v cukrárně. Myslíte, že v cukrárně mají Dinicovy řezy? Myslím, že v cukrárně je většina řezů minimální. *(odposlechnuto na přednášce)*

[5] Princip holubníku a nějaká ta ±1.

Po k krocích máme alespoň k vrstev, a proto existuje vrstva δ s nejvýše n/k vrcholy. Tedy existuje řez C o velikosti $|C| \leq n/k$ (získáme z vrstvy δ výše popsaným postupem). Algoritmu zbývá do konce $\leq n/k$ iterací vnějšího cyklu, celkem tedy udělá $k + n/k$ iterací. Nyní stačí zvolit $k = \sqrt{n}$ a složitost celého algoritmu vyjde $\mathcal{O}(\sqrt{n} \cdot m)$.

Mimochodem, hledání maximálního párování pomocí Dinicova algoritmu je také ekvivalentní známému Hopcroft-Karpově algoritmu [1]. Ten je založen na střídavých cestách z předchozí kapitoly a v každé iteraci nalezne množinu vrcholově disjuktních nejkratších střídavých cest, která je maximální vzhledem k inkluzi. Touto množinou pak aktuální párování přexoruje, čímž ho zvětší. Všimněte si, že tyto množiny cest odpovídají právě blokujícím tokům v pročištěné síti rezerv, takže můžeme i zde použít náš odhad na počet iterací.

Třetí pokus pro jednotkové kapacity bez omezení na stupně vrcholů v síti: Hlavní myšlenkou je opět po k krocích najít nějaký malý řez. Najdeme dvě malé sousední vrstvy a všechny hrany mezi nimi budou tvořit námi hledaný malý řez. Budeme tentokrát předpokládat, že naše síť není multigraf, případně že násobnost hran je alespoň omezena konstantou.

Označme s_i počet vrcholů v i-té vrstvě. Součet počtu vrcholů ve dvou sousedních vrstvách označíme $t_i = s_i + s_{i+1}$. Bude tedy platit nerovnost:

$$\sum_i t_i \leq 2 \sum_i s_i \leq 2n.$$

Podle holubníkového principu existuje i takové, že $t_i \leq 2n/k$, čili $s_i + s_{i+1} \leq 2n/k$. Počet hran mezi s_i a s_{i+1} je velikost řezu C, a to je shora omezeno $s_i \cdot s_{i+1}$. Nejhorší případ nastane, když $s_i = s_{i+1} = n/k$, a proto $|C| \leq (n/k)^2$. Proto počet iterací velkého cyklu je $\leq k + (n/k)^2$. Chytře zvolíme $k = n^{2/3}$. Složitost celého algoritmu pak bude $\mathcal{O}(n^{2/3}m)$.

Obecný odhad pro celočíselné kapacity: Tento odhad je založen na velikosti maximálního toku f a předpokladu celočíselných kapacit. Za jednu iteraci velkého cyklu projdeme malým cyklem maximálně tolikrát, o kolik se v něm zvedl tok, protože každá zlepšující cesta ho zvedne alespoň o 1. Zlepšující cesta se tedy hledá maximálně $|f|$-krát za celou dobu běhu algoritmu. Cestu najdeme v čase $\mathcal{O}(n)$. Celkem na hledání cest spotřebujeme $\mathcal{O}(|f| \cdot n)$ za celou dobu běhu algoritmu.

Nesmíme ale zapomenout na čištění. V jedné iteraci velkého cyklu nás stojí čištění $\mathcal{O}(m)$ a velkých iterací je $\leq n$. Proto celková složitost algoritmu činí $\mathcal{O}(|f|n + nm)$

Pokud navíc budeme předpokládat, že kapacita hran je nejvýše C a G není multigraf, můžeme využít toho, že $|f| \leq Cn$ (omezeno řezem okolo s) a získat odhad $\mathcal{O}(Cn^2 + nm)$.

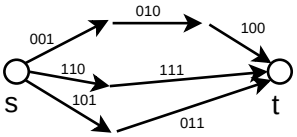
Škálování kapacit

Pokud jsou kapacity hran větší celá čísla omezená nějakou konstantou C, můžeme si pomoci následujícím algoritmem. Jeho základní myšlenka je podobná, jako

u třídění čísel postupně po řádech pomocí radix-sortu neboli přihrádkového třídění. Pro jistotu si ho připomeňme. Algoritmus nejprve setřídí čísla podle poslední (nejméně významné) cifry, poté podle předposlední, předpředposlední a tak dále.

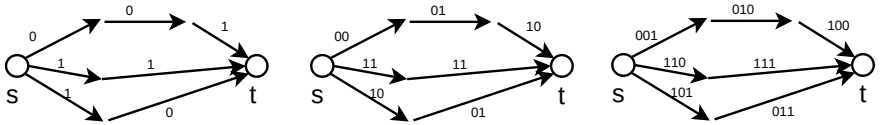
V našem případě budeme postupně budovat sítě čím dál podobnější zadané síti a v nich počítat toky, až nakonec získáme tok pro ni.

Přesněji: Maximální tok v síti G budeme hledat tak, že hranám postupně budeme zvětšovat kapacity bit po bitu v binárním zápisu až k jejich skutečné kapacitě. Přitom po každém posunu zavoláme Dinicův algoritmus, aby dopočítal maximální tok. Pomocí předchozího odhadu ukážeme, že jeden takový krok nebude příliš drahý.



Původní síť, na hranách jsou jejich kapacity v binárním zápisu

Označme k index nejvyššího bitu v zápisu kapacit v zadané síti ($k = \lfloor \log_2 C \rfloor$). Postupně budeme budovat sítě G_i s kapacitami $c_i(e) = \lfloor c(e)/2^{k-i} \rfloor$. G_0 je nejořezanější síť, kde každá hrana má kapacitu rovnou nejvyššímu bitu v binárním zápisu její skutečné kapacity, až G_k je původní síť G.



Sítě G_0, G_1 a G_2, jak vyjdou pro síť z předchozího obrázku

Přitom pro kapacity v jednotlivých sítích platí:

$$c_{i+1}(e) = \begin{cases} 2c_i(e), & \text{pokud } (k-i-1)\text{-tý bit je 0,} \\ 2c_i(e)+1, & \text{pokud } (k-i-1)\text{-tý bit je 1.} \end{cases}$$

Na spočtení maximálního toku f_i v síti G_i zavoláme Dinicův algoritmus, ovšem do začátku nepoužijeme nulový tok, nýbrž tok $2f_{i-1}$. Rozdíl toku z inicializace a výsledného bude malý, totiž:

Lemma: $|f_i| - |2f_{i-1}| \leq m$.

Důkaz: Vezmeme minimální řez R v G_{i-1}. Podle F-F věty víme, že $|f_{i-1}| = |R|$. Řez R obsahuje $\leq m$ hran, a tedy v G_i má tentýž řez kapacitu maximálně $2|R| + m$. Maximální tok je omezen každým řezem, tedy i řezem R, a proto tok vzroste nejvýše o m. ♡

Podle předchozího odhadu pro celočíselné kapacity výpočet toku f_i trvá $\mathcal{O}(mn)$. Takový tok se bude počítat k-krát, pročež celková složitost vyjde $\mathcal{O}(mn \log C)$.

Algoritmus tří Indů

Překvapení na konec: Dinicův algoritmus lze poměrně snadno zrychlit i ve zcela obecném případě. Malhotra, Kumar a Maheshwari vymysleli efektivnější algoritmus [2] na hledání blokujícího toku ve vrstevnaté síti, který běží v čase $\mathcal{O}(n^2)$ a použijeme-li ho v Dinicově algoritmu, zrychlíme hledání maximálního toku na $\mathcal{O}(n^3)$. Tento algoritmus vešel do dějin pod názvem Metoda tří Indů.

Mějme tedy nějakou vrstevnatou síť. Začneme s nulovým tokem a budeme ho postupně zlepšovat. Průběžně si budeme udržovat rezervy hran $r(e)$[6] a také následující rezervy vrcholů:

Definice: $r^+(v)$ je součet rezerv všech hran vstupujících do v, $r^-(v)$ součet rezerv hran vystupujících z v a konečně $r(v) := \min(r^+(v), r^-(v))$.

V každé iteraci algoritmu nalezneme vrchol s nejnižším $r(v)$ a zvětšíme tok tak, aby se tato rezerva vynulovala. Za tímto účelem nejdříve přepravíme $r(v)$ jednotek toku ze zdroje do v: u každého vrcholu w si budeme pamatovat *plán* $p(w)$, což bude množství tekutiny, které potřebujeme dostat ze zdroje do w. Nejdříve budou plány všude nulové až na $p(v) = r(v)$. Pak budeme postupovat po vrstvách směrem ke zdroji a plány všech vrcholů splníme tak, že je převedeme na plány vrcholů v následující vrstvě, až doputujeme ke zdroji, jehož plán je splněn triviálně. Nakonec analogickým způsobem protlačíme $r(v)$ jednotek z v do spotřebiče.

Během výpočtu průběžně přepočítáváme všechna r^+, r^- a r podle toho, jak se mění rezervy jednotlivých hran (při každé úpravě rezervy to zvládneme v konstantním čase) a síť čistíme stejně jako u Dinicova algoritmu.

Algoritmus: (hledání blokujícího toku ve vrstevnaté síti podle tří Indů)

1. $f_B \leftarrow$ *prázdný tok*.
2. Spočítáme rezervy všech hran a r^+, r^- a r všech vrcholů. (Tyto hodnoty budeme posléze udržovat při každé změně toku po hraně.)
3. Dokud v síti existují vrcholy s nenulovou rezervou, vezmeme vrchol v s nejmenším $r(v)$ a provedeme pro něj: *(vnější cyklus)*
4. Převedeme $r(v)$ jednotek toku z s do v následovně:
5. Položíme $p(v) \leftarrow r(v)$, $p(\cdot) = 0$.
6. Procházíme vrcholy sítě po vrstvách od v směrem k s. Pro každý vrchol w provedeme:
7. Dokud $p(w) > 0$:
8. Vezmeme libovolnou hranu uw a tok po ní zvýšíme o $\delta = \min(r(uw), p(w))$. Tím se $p(w)$ sníží o δ a $p(u)$ zvýší o δ.
9. Pokud se hrana uw nasytila, odstraníme jí ze sítě a síť dočistíme.
10. Analogicky převedeme $r(v)$ jednotek z v do t.

[6] počítáme pouze rezervu ve směru hrany, neboť nám stačí najít blokující tok, ne nutně maximální

Analýza: Nejprve si všimneme, že cyklus v kroku 8 opravdu dokáže vynulovat $p(w)$. Součet všech $p(w)$ přes každou vrstvu je totiž nejvýše roven $r(v)$, takže speciálně každé $p(w) \leq r(v)$. Jenže $r(v)$ jsme vybrali jako nejmenší, takže $p(w) \leq r(v) \leq r(w) \leq r^+(w)$, a proto je plánovaný tok kudy přivést. Proto se algoritmus zastaví a vydá blokující tok.

Zbývá odhadnout časovou složitost: Když oddělíme převádění plánů po hranách (kroky 7–9), zbytek jedné iterace vnějšího cyklu trvá $\mathcal{O}(n)$ a těchto iterací je nejvýše n. Všechna převedení plánu si rozdělíme na ta, kterými se nějaká hrana nasytila, a ta, která skončila vynulováním $p(w)$. Těch prvních je $\mathcal{O}(m)$, protože každou takovou hranu vzápětí odstraníme a čištění, jak už víme, trvá také lineárně dlouho. Druhý případ nastane pro každý vrchol nejvýše jednou za iteraci. Dohromady tedy trvají všechna převedení $\mathcal{O}(n^2)$, stejně jako zbytek algoritmu. ♡

Přehled variant Dinicova algoritmu

varianta	*čas*
standardní	$\mathcal{O}(n^2m)$
jednotkové kapacity	$\mathcal{O}(\sqrt{m} \cdot m) = \mathcal{O}(m^{3/2})$
jednotkové kapacity, 1 stupeň ≤ 1	$\mathcal{O}(\sqrt{n} \cdot m)$
jednotkové kapacity, prostý graf	$\mathcal{O}(n^{2/3}m)$
celočíselné kapacity	$\mathcal{O}(\lvert f\rvert \cdot n + nm)$
celočíselné kapacity $\leq C$	$\mathcal{O}(Cn^2 + mn)$
celočíselné kapacity $\leq C$ (škálování)	$\mathcal{O}(mn \log C)$
tři Indové	$\mathcal{O}(n^3)$

Literatura

[1] J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[2] V. Malhotra, M. Kumar, and S. Maheshwari. An $\mathcal{O}(\lvert V\rvert^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, 7(6):277–278, 1978.