1 Toky v sítích

Jednotlivé sekce této kapitoly (algoritmy Ford-Fulkersonův, Dinitzův a Goldbergův) jsou rozděleny do tří částí:

- **Procházka** je návod k použití příslušného appletu Algovize a zárove vysvětlením myšlenky, na které je daný algoritmus založen intuitivním způsobem za pomoci pohyblivých obrázků. Někdy dokonce může být výklad trochu nepřesný nebo neúplný, pokud by snaha o přesnost a úplnost byla na závadu srozumitelnosti výkladu. Vše si ale nahradíme více než dostatečně v dalších dvou částech sekce.
- **Laboratoř** - tim je míněna počítačová laboratoř, kam člověk přijde, aby naklepal text programu, implementujícího algoritmus, do počítače a po odladění spustil a provozoval. V této části jsou vypsány příslušné programy způsobem, který kopíruje syntax procedurálních jazyků, ale řadu detailů (například volba a manipulace s datovými strukturami jako je třeba množina, fronta, seznam vrcholů nebo hran) popisuje slovně, spoléhajíce na to, že čtenář a uživatel textů již není v programování úplný začátečník.
- **Škola je místo**, kde se zcela korektním způsobem učíme, proč a jak něco je. V tomto případě proč je program popsaný v laboratoři správný (tedy spočítá to, co má spočítat), proč se zastaví a za jak dlouho. Text sice vypadá jako matematika (je členěn do posloupnosti definic, lemmat a vět), ale nelekejte se. Nechci vás učit matematiku, jen použít exaktní způsob uvažování, který je vlastní matematikům a měl by být vlastní i dobrým programátorům, v tomto případě jde uvažování o kousku software. Ve skutečnosti se jedná o přístum, který je nejbližší nikoliv teoretické disciplině, ale praktickému softwarovému přístupu, označovaném jako verifikace programů (nebo podobně). V něm jde o formální stanovení specifikací toho, co program má vykonávat a pak o stejně formální verifikaci toho, že daný kód tyto specifikace spluje. Pokud se vám zdá text ve škole příliš formální, nezoufejte. Skutečná verifikace praktického programu je vždy daleko detailnější, formálnější a tedy pro člověka i nudnější a přitom obvykle tak obsáhlá, že se ji většinou snažíme dělat přinejmenším s částečnou podporou počítače.

Celý algoritmus je tedy vlastně popsán dvakrát: jednou během procházky appletem, kdy je výklad hodně intuitivní a má za cíl vyložit myšlenku, která za algoritmem stojí, a podruhé v laboratoři a ve škole, kdy je cílem přesnost a úplnost a názornost jde stranou, spoléhajíce na to, že už víte o co jde s procházky a tudíž se i ve formálně psaném textu orientujete.

2 Ford-Fulkersonův algoritmus

3 Procházka

Scéna: *Největší tok*

Tato scéna ukazuje *síť*, což je orientovaný graf spolu se dvěma jeho vrcholy, které označujeme jako *zdroj* (zobrazen zeleně) a *spotřebič* (zobrazen modře). Kromě toho je každé hraně grafu přiřazeno nezáporné reálné číslo, nazývané *kapacita* hrany. Vrcholy sítě různé od zdroje a spotřebiče nazýváme *vnitřní vrcholy*.

Tok v síti je funkce t, která každé hraně h přiřazuje číslo $t(h)$, pro které platí $0 \leq t(h) \leq c(h)$, kde $c(h)$ je kapacita hrany h a navíc pro každý vnitřní vrchol v platí, že součet toků hranami vstupujícími do v je roven součtu toků hranami vystupujícími z v, tedy

$$\sum_{h \in in(v)} t(h) = \sum_{k \in out(v)} t(k),$$

kde $in(v)$, resp. $out(v)$, je množina hran sítě vstupujících do v, resp. vystupujících z v.

Je možno si představit, že uzly jsou města, hrany jsou úseky železnice nebo silnice, kterými se snažíme dopravovat jistý druh zboží z jednoho města (zdroj) do jiného města (spotřebič), přičemž každá cesta má omezenou průchodost a zboží se skutečně jen přepravuje, ale v žádném jiném než výchozím nebo cílovém městě se neskladuje, nespotřebovává ani neztrácí, ale také nevzniká.

Je také možno si hrany představovat jako vodovodní nebo ropná potrubí nebo elektrická vedení, kdy je požadavek bezeztrátové dopravy obvykle přirozeným způsobem splněn.

V rámci výše uvedených omezení se budeme snažit o přepravu co největšího množství zboží nebo tekutiny ze zdroje. Toto množství je dáno přebytkem toku ve spotřebiči (kam obvykle tok pouze přitéká, i když definice nezabraňuje tomu, aby část byla současně odváděna a třeba i vedena zpět do zdroje) a je současně rovno deficitu zdroje (ze kterého obvykle tok jen odtéká).

Největší tok pro zobrazenou síť je možno zobrazit stisknutím knoflíku [**Krok**] (a vrátit se do výchozího stavu knoflíkem [**Zpět**]). Před provedením výpočtu toku je v této scéně u každé hrany uvedena její kapacita; po výpočtu toku je u ní uvedena dvojice tok:kapacita (a hrany, kterými teče nulový tok zešednou). Tak tomu je, pokud je na ovladači zvoleno [**Automatické označení**]. Změnou volby je možno dosáhnout toho, že je stále zobrazena kapacita nebo stále zobrazen tok a nebo je stále zobrazena dvojice čísel tok:kapacita. Ověřte si, že pro vypočtený tok platí výše uvedené podmínky.

Scéna umožňuje běžné úpravy sítě jako přidávání, odebírání a přesouvání vrcholů a přidávání a odebírání hran a změny jejich kapacity (po klepnutí na hranu pravým knoflíkem myši a volbě v popup menu). Je také možno změnit zdroj i spotřebič (po klepnutí na zvolený vrchol pravým knoflíkem myši vybrat v popup menu) nebo zobrazit některý z předdefinovaných příkladů. K editaci je třeba kliknout na knoflík [**Více ovladačů**] a pak volbu [**Počítej**] změnit na [**Edituj**].

Scéna: *Hladový algoritmus*

Základní myšlenka Ford-Fulkersonova algoritmu je jednoduchá: začínáme nulovým tokem (žádnou hranou nic neteče), který postupně zvětšujeme tak dlouho, dokud nenalezneme největší možný tok. Nejjednodušší (ale, jak ukážeme dále, nedostatečný) způsob zvětšení toku je následující: nalezneme orientovanou cestu ze zdroje do spotřebiče takovou, že každou její hranou protéká méně toku než je její kapacita, takže je možné tok ještě o něco zvýšit, a pak zvýšíme hodnotu toku všemi hranami o jistou hodnotu Δ.

Hranami cesty musí protékat tok menší než jejich kapacity, aby tok bylo možno zvětšit bez porušení kapacitního omezení. Proto tok hranou h s kapacitou $c(h)$, kterou již teče tok $t(h)$, je možné zvýšit nejvýše o $c(h) - t(h)$. V důsledku toho největší hodnota Δ, která neporuší kapacitní omezení, je rovna $\min_h(c(h) - t(h))$, kde minimum se bere přes hrany cesty. Kirchhofův zákon (rovnost přítoku a odtoku ve všech vnitřních vrcholech sítě) se neporuší, protože pro každý vrchol cesty různý od zdroje a spotřebiče se přítok i odtok zvýší o Δ. Velikost toku vycházejícího ze zdroje se ovšem zvýší o Δ.

Krokujte si algoritmus a sledujte hledání a zpracování cest. Na ovladači je možno nastavit, jak detailně se krokování bude provádět a to odděleně pro fázi hledání cesty

Uvedený algoritmus se obvykle nazývá hladový, protože jen stále ujídá kapacitu hran pro zvýšení toku a není ochoten tok některou hranou snížit pro dosažení globálního zlepšení.

Výpočet probíhá pomalu, ale pro pochopení myšlenky algoritmu stačí jej sledovat jen chvíli.

Bohužel se může stát, jak uvidíme v příští scéně, že další zlepšení toku využitím přímé zlepšující cesty není možné, ale optimální tok přesto není nalezen.

V této a mnoha následujících scénách se také ukazuje pseudokód algoritmu, ve kterém je znázorněno, jak výpočet probíhá. Pokud vám pseudokód vadí, vypněte jeho zobrazování (je-li třeba, zvolte [**Více ovladačů**] a pak změňte [**Program automaticky**] na [**Skryj program**].

Scéna: *Hladový algoritmus - selhání*

V této scéně je ukázána jedna velmi jednoduchá síť, ve které postupem podle předchozí scény nalezneme tok, který již hladový algoritmus dále zlepšit neumí, ale *není* největší možný, neboť v síti existuje odlišný tok větší velikosti. Je z toho vidět, že neustálé zvyšování toku použitím přímé zlepšující cesty může vést do slepé uličky a, jak uvidíme dále, je občas nutno přikročit k tomu,

že se tok některými hranami třeba i opakovaně sníží a pak se začne zvyšovat jiným způsobem a teprve tak bude možno nalézt optimální tok.

U zobrazeného příkladu optimální tok využívá plné kapacity hran na obvodu sítě, ale nevyužívá příčku přes střed sítě; jeho velikost je 2. Výpočetní postup je volen tak, že ale hladový algoritmus nalezne tok velikosti 1, který již nelze zlepšit pouhým zvětšováním toků hranami. Je to proto, že při zpracování první nalezené cesty se zvýší tok příčkou na 1, ale snadno si můžete ověřit, že žádný tok, ve kterém příčkou teče nenulová hodnota toku, nemůže mít optimální velikost 2. Přitom hladový algoritmus neumí tok příčkou zmenšit.

Volbu zlepšujících cest provádí Algovize sama, protože je možno ukázat, že v libovolné síti lze hladový algoritmus navádět (t.j volit cesty pro zlepšení toku) takovým způsobem, že optimální tok najde. Například v našem příkladu by stačilo napřed vybrat cestu složenou ze dvou horních hran a pak cestu obsahující spodní dvě hrany. Mohlo by se proto stát, že uživatel bude záměrně nebo náhodou volit cesty tak, že nalezne optimální tok, zatímco cílem scény je ukázat, že algoritmus může zabloudit a zastavit se, aniž by nalezl největší možný tok.

Volbu cest pro nalezení optimálního toku hladovým způsobem by bylo možno poměrně snadno provést zpětně ze znalosti výsledku výpočtu. Není ale známa žádná metoda, která by takovou volbu zaručila bez předběžné znalosti výsledku a proto hladový algoritmus nemůže zaručit optimalitu nalezeného řešení.

Scéna: *Přenos přebytku*

V následujících několika scénách bude ukázán jiný pohled na zlepšování toku, který umožní upravit hladový algoritmus tak, aby bylo optimální řešení vždy nalezeno. Toto zlepšení popsali v roce 1964 Ford s Fulkersonem, po nichž je zlepšený algoritmus nazván.

Představme si, že tok hranami cesty zvětšujeme o určenou hodnotu Δ postupně v tom pořadí, v jakém se na cestě nacházejí. Pak je v průběhu této akce vždy v jednom vrcholu (koncovém vrcholu naposledy zpracované hrany) porušena podmínka rovnosti toku přitékajícího a toku odtékajícího - do vrcholu přitéká o Δ jednotek toku více než před zpracováním cesty, ale odtéká zatím stejně. Tento rozdíl budeme nazývat přebytek.

Využívajíce analogii s vodovodní sítí budeme přebytek znázorňovat výší barevného sloupečku nad vrcholem, představujícího vyrovnávací nádrž s proměnlivou výškou hladiny, který je úměrná rozdílu přítoku do vrcholu a odtoku z vrcholu. Jestliže je přebytek záporný (tedy ve skutečnosti deficit - nastává to pouze u zdroje), bude sloupeček pod vrcholem představovat hladinu ve studni, ze které rozdíl dočerpáváme.

Zkuste si knoflíkem [**Krok**] krokovat průběh výpočtu zpracování první cesty. Algovize nejprve najde (hranu po hraně) cestu ze zdroje do spotřebiče a určí patřičné Δ. Potom, opět po jednotlivých hranách, zvyšuje tok. Hrana, ve které se bude tok zvyšovat je znázorněna červeně; je vidět, že zvýšení toku hranou má za následek odvedení přebytku z jejího počátečního vrcholu do jejího koncového vrcholu. V okamžiku, kdy se kladný přebytek dostane až do spotřebiče (ve kterém je přípustný), zpracování cesty končí a začíná určení a zpracování další cesty (pokud existuje).

Algoritmus pracuje podobně jako v předchozí scéně - zvolí si nejprve cestu přes příčku, takže pokud by pracoval hladově, po zpracování první cesty by se zastavil, aniž by byl nalezen optimální tok. Zde ale výpočet pokračuje dále. Místo aby nalezl další orientovanou cestu ze zdroje do spotřebiče, Algovize ukáže postupně tři hrany, z nichž prostřední je ale v protisměru.

Pak se zahájí přenos přebytku ze zdroje do spotřebiče. Nejprve se již známým způsobem odvede další jednotka toku za zdroje do spodního vrcholu zvýšením toku příslušnou hranou. Pak se střední příčka zobrazí jako široká šipka; její šířka odpovídá kapacitě a šířka červeného pruhu odpovídá toku, který jí prochází.

Jelikož v tomto okamžiku širokou hranou prochází tok velikosti 1, který je roven její kapacitě, je červený pruh roztažen přes celou její šířku. Přejeďte ale myší se stisknutým knoflíkem přes hranu. Hrana červeného pásu sleduje kurzor, takže se jeho šířka mění a tím je možno měnit velikost toku hranou z maximální hodnoty až na nulu.

Snižování toku příčkou (tok teče shora dolů) způsobí, že se přebytek ze spodního vrcholu příčky snižuje a o tutéž hodnotu se ale zvyšuje přebytek horního vrcholu příčky (původně nulový).

Přebytek je tedy také možno převést z jednoho vrcholu do druhého tak, že se *sníží tok hranou v protisměru*, pokud jí teče nenulový tok.

V původní verzi proto Ford-Fulkersonův algoritmus zlepšoval tok podle tak zvaných zobecněných cest, na kterých byly hrany ve směru s rezervou a hrany v protisměru s nenulovým tokem. Než ukážeme, že takto sestrojený algoritmus již *vždy* nalezne optimální tok, zavedeme si v následujících několika scénách pojmy, které umožňují formulovat algoritmus jednotným a zjednodušeným způsobem, bez neustálého rozdělování hran zobecněné cesty na přímé a zpětné.

Scéna: *Tok hranou*

Abychom nemuseli stále rozebírat, zda k přenosu přebytku dochází hladovým zvyšování toku hranou nebo snižováním toku protisměrnou hranou, zavedeme v několika následujících jednoduchých scénách způsob práce s přebytky, který situaci zjednoduší.

Tato scéna je jen přípravná a graficky znázorňuje toky ve dvou opačně orientovaných hranách mezi dvěma vrcholy. Tahnutím myši uvnitř jedné z těchto širokých hran se mění šířka pásu, který graficky znázorňuje velikost toku hranou (v horní hraně, jdoucí zleva doprava, je světle zelený, v dolní hraně směřující vlevo, je tmavě zelený). Šířka hrany odpovídá její kapacitě. Zkuste v obou hranách nastavit nenulový tok. Žluté sloupce u konců hran znázorňují přebytky toku v koncových vrcholech hran (nebo přesněji příspěvky k přebytkům toku od zobrazených hran). Přebytek je kladný, pokud je sloupec nad dělící čárou mezi vyobrazenými hranami, záporný pokud je sloupec pod čárou. Měňte toky a sledujte, že vždy jeden přebytek roste a druhý o tutéž hodnotu klesá.

Scéna: *Cirkulace*

Scéna je v zásadě opakováním předchozí. Obecná definice i náš applet připouští, aby oběma opačnými hranami tekl kladný tok. Z praktického hlediska i z hlediska hledání největšího toku to ale není příliš účelné; vozíme-li jisté zboží z Prahy do Brna a současně stejné zboží vezeme z Brna do Prahy, jen plýtváme prostředky.

Ta část toku, která teče v obou hranách současně, je na obrazovce znázorněna modře a nazýváme ji cirkulace. (Pro sečtělé čtenáře: náš pojem cirkulace je jen speciálním případem toho, co se obvykle cirkulací také nazývá).

Jestliže cirkulaci odečteme od toků oběma hranami, naše situace je může jen zlepšit: Toky se sníží při zachování jejich nezápornosti, takže kapacitní omezení zůstanou v platnosti, ale hrany mají více rezervy pro další zvyšování toku. Současné snížení toku v obou hranách o stejnou hodnotu také nezmění přebytky ve vrcholech a proto tok zůstane tokem (nulové přebytky ve vnitřních vrcholech sítě) a jeho velikost (přebytek ve spotřebiči) se také nezmění. Z toho plyne, že existuje maximální tok (náš cíl), bez cirkulací.

Scéna: *Tok bez cirkulace*

Tato scéna se podobá předchozí, ale ukazuje již toky bez cirkulací. Jesliže tedy v jedné z obou hran je nenulový tok, tok druhou je nulový. Zkuste si opět nastavovat toky hranami; požadavek nenulového toku jednou hranou vynuluje tok v druhé z nich.

Se stisknutou myší se nyní pohybujte od horního okraje horní hrany ke spodnímu okraji dolní hrany. Je zřejmé, že okamžitý stav (toky hranami) je popsán jediným parametrem - výškou rozhraní taženého myší - který se mění v rozmezí od vrchního okraje horní hrany po spodní okraj dolní hrany a někdy představuje tok zleva doprava, jindy tok směrem opačným. Jak rozhraní stoupá, zvyšuje se přebytek v pravém vrcholu a současně klesá přebytek levého vrcholu.

Je snadno vidět, že rezerva pro zvýšení přebytku vrcholu vpravo (a současně pro snížení přebytku vlevo) je dána vzdáleností horního okraje horní hrany a pohyblivého rozhraní, jak je naznačeno červenou kótou s popisem "rezerva" umístěným nad hranami. Pokud je tok horní hranou kladný, pak tato rezerva je rovna její kapacitě, snížené o velikost toku hranou; rezerva je tudíž menší než její kapacita. Jestliže ale je kladný tok spodní hranou, který jde v opačném směru, pak přebytek zleva doprava lze přenášet tak, že se nejprve snižuje až na nulu tok spodní hranou a pak se zvyšuje tok horní hranou až do její kapacity. Rezerva pro přenos přebytku je tedy *větší* než kapacita horní hrany. Obdobně tomu je i se spodní hranou.

Scéna: *Rezerva*

Tato scéna opakuje předchozí, ale je v ní vynechána hranice mezi hranami, aby bylo zdůrazněno, že možnost zvyšování nebo snižování přebytků vrcholů nezávisí na této hranici, která určuje kapacity hran, ale pouze na vzdálenosti mezi horním okrajem horní hrany a dolním okrajem dolní hrany, který je dán součtem kapacit těchto hran. Rezerva pro zvýšení přebytku pravého vrcholu, daná vzdáleností horního okraje horní hrany a pohyblivého rozhraní (šířka bílého pásu) a rezerva pro zvýšení přebytku levého vrcholu, daná vzdáleností pohyblivého rozhraní a dolního okraje dolní hrany (šířka šedého pásu), jsou pak určeny polohou pohyblivého rozhraní. Součet těchto rezerv pro zvýšení přebytku je zjevně konstantní a rovný součtu kapacit hran.

Pro úplný popis situace proto bude vždy (předpokládajíce znalost kapacit hran) dostatečné znát rezervu jedné z hran (druhou lze snadno dopočítat).

Scéna: *Tok a rezerva*

Pro popis algoritmů je daleko výhodnější používat rezervy hran a ne jejich toky, protože přenos přebytku podél hrany e se v řeči toků musí rozdělovat do dvou možností: zvýšení toku hranou e nebo snížení toku hranou opačnou, zatímco v řeči rezerv jde v obou případech o snížení rezervy hrany e (za současného zvýšení rezervy hrany opačné o stejnou hodnotu, protože součet těchto rezerv je konstantní a rovný součtu kapacit hran).

Za předpokladu nulového výchozího toku se proto na počátku výpočtu položí výchozí rezervy hran rovné jejich kapacitám a tok se upravuje tak, že se přenáší podél hran přebytky ve vrcholech za současné změny rezerv dotčených hran. Na konci výpočtu ale je nutné zpět z rezerv hran spočítat toky jdnotlivými hranami. Tato scéna ukazuje, jak se to provede.

Nastavte pomocí myši vhodně rezervy jako v předchozí scéně (t.j. zvolte polohu rozhraní mezi bílým pásem, který je rezervou pro přenos zleva doprava a šedým pásem - rezerva zprava doleva) a klikněte na checkbox [**Výpočet toku**]. Absolutní velikost toku se zobrazí jako šířka zeleného pásu a směr toku, t.j. jedná-li se o tok hranou zleva doprava nebo zprava doleva, se pozná podle zkosení daného čelem odpovídající hrany.

Jednoduchý program pro výpočet toků z rezerv si ukážeme v laboratoři, ale možná si ho už teď napíšete sami.

Scéna: *Nulová kapacita*

Nyní ještě závěrečná scéna týkající se toků a kapacit: spodní hrana zde má nulovou kapacitu (což je vlastně totéž jako by v síti nebyla). Z obrázku je ale vidět, že tato hrana může i přesto mít nenulovou kladnou rezervu, což ostatně plyne i z matematické definice rezervy; ta je v takovém případě rovna toku opačnou hranou a znamená, že změna přebytků koncových vrcholů se dá provést snížením toku opačnou hranou.

V následujících scénách týkajících se toků v sítích budeme pro formulaci algoritmů používat výhradně pojmu rezervy hrany a toky hranami se budou dopočítávat až dodatečně na konci výpočtu.

Ve většině scén budeme používat zobrazení ukazující ty hrany, které mají kladnou rezervu (včetně těch, které byly do sítě doplněny dodatečně s nulovou kapacitou), ale na druhé straně *skrývající* hrany s nulovou rezervou, protože ty již nejsou použitelné k dalšímu přenosu přebytku. (Týká se volby [**rezerva**] a [**rezerva:kapacita**] - v některých scénách a v módu [**Práce**] lze způsob zobrazení volit z více možností).

Nakonec shrneme a trochu rozvineme to, co bylo vysvětleno v této a několika předchozích scénách. Předpokládejme, že je na hranách sítě definována jistá funkce t, která sice vyhovuje stejným kapacitním omezením jako tok, ale může mít nezáporný přebytek i ve vnitřních vrcholech (tedy je povoleno, aby do vnitřních vrcholů přitékalo hranami více než odtéká). Takové funkci budeme říkat zobecněný tok. Speciálně tedy t může být tok, tak jak byl definován v první scéně. Pak řekneme, že rezerva uspořádané dvojice $h = (u, v)$ (která nemusí být hranou sítě) je číslo $r(h)$, definované takto:

- jestliže h je hranou sítě, ale h^{op} hranou není, pak $r(h) = c(h) - t(h)$;

- jestliže h^{op} je hranou sítě, ale h hranou není, pak $r(h) = t(h^{op})$;
- jestliže h i h^{op} jsou hranami sítě a $t(h^{op}) = 0$, pak $r(h) = c(h) - t(h)$;
- jestliže h i h^{op} jsou hranami sítě a $t(h) = 0$, pak $r(h) = c(h) + t(h^{op})$;
- jestliže ani h ani h^{op} nejsou hranami sítě, pak $r(h) = 0$;

kde h^{op} představuje opačnou dvojici (v, u). Rozpomeňte se také na naši úmluvu, že nebudeme připouštět, aby (pokud h i h^{op} jsou hranami sítě) platilo současně $t(h) > 0$ a $t(h^{op}) > 0$.

Pokud má dvojice $h = (u, v)$ kladnou rezervu R a pokud t je zobecněný tok, který má ve vrcholu u kladný přebytek E, pak pokud Δ je kladné číslo takové, že $\Delta < min(R, E)$, je možno manipulací s hodnotami t pro hranu $h = (u, v)$ a/nebo $h^{op} = (v, u)$ (pokud jsou hranami sítě) dosáhnout snížení přebytku t ve vrcholu u o hodnotu Δ za současného zvýšení přebytku vrcholu v o tutéž hodnotu Δ. Přitom dojde současně k tomu, že rezerva hrany h o Δ poklesne a současně se rezerva opačné hrany h^{op} o Δ zvýší. Této operaci budeme říkat přenesení přebytku velikosti Δ z vrcholu u do vrcholu v.

Konkrétně manipulace s toky v h a/nebo h^{op} při přenášení přebytku z vrcholu u do vrcholu v vypadá takto:

- jestliže h^{op} je hranou sítě a $0 < \Delta < t(h^{op})$, pak hodnotu $t(h^{op})$ snížíme o Δ;
- jestliže h^{op} je hranou sítě a $0 < t(h^{op}) < \Delta$, položíme $\Delta_1 = \Delta - t(h^{op})$ a nejprve hodnotu $t(h^{op})$ snížíme na nulu (tedy o $\Delta - \Delta_1$) a pak hodnotu $t(h)$ zvýšíme o Δ_1 (ve škole si ukážeme, že h v tomto případě musí být hranou);
- jestliže h^{op} není hrana nebo $t(h^{op}) = 0$, pak hodnotu $t(h)$ zvýšíme o Δ;

Nakonec si povšimněte, že pokud t je nulový tok, pak rezervy hran sítě jsou rovny jejich kapacitám, kdežto rezervy pro dvojice vrcholů, které nejsou hranami, jsou nulové.

Scéna: *Ford-Fulkersonův algoritmus*

Jak už bylo naznačeno výše, Ford-Fulkersonův algoritmus je s využitím pojmu rezervy hrany možno popsat velmi jednoduše. Místo práce v původní síti budeme pracovat v síti, kterou budeme nazývat síť rezerv. Síť rezerv má stejné vrcholy jako původní síť, ale množina jejích hran závisí na tom, jaký v základní síti teče tok nebo zobecněný tok popsaný v prvním odstavci popisu této scény a proto se v průběhu výpočtu stále mění. Předpokládáme-li, že v jistý okamžik teče základní sítí zobecněný tok t, pak hranami v síti rezerv právě ty dvojice $h = (u, v)$, které mají kladnou rezervu, tedy pro které platí $r(h) > 0$.

Algoritmus začíná se z nulového toku a dokud je možné nalézt cestu ze zdroje do spotřebiče, která je tvořena hranami s kladnými rezervami, pak jednu z takových cest vybereme a po jejích hranách přesuneme ze zdroje do spotřebiče přebytek rovný minimu Δ z rezerv hran cesty. Tato hodnota je největší možná velikost přebytku, který se dá postupně hranami přenést, aniž by došlo k porušení kapacitních omezení. Je zřejmé, že touto operací se velikost toku zvýší o Δ.

Zkuste si výpočet pro zobrazený příklad a případně i další sítě z nabízeného souboru nebo vlastní konstrukce. Na displeji je (s výjimkou začátku a konce výpočtu) zobrazována síť rezerv, tedy šipky představují dvojice vrcholů, které mají kladnou rezervu. Je velmi důležité, abyste pochopili, jak se při zpracování jedné cesty změní rezervy hran a jak se změní množina hran sítě rezerv: všem hranám zpracovávané cesty poklesne rezerva o Δ. Alespoň jedné hraně cesty (ale často i více, například všem hranám cesty) poklesne rezerva na nulu a tím pádem ze sítě rezerv vypadne. Zároveň se všem opačným hranám rezerva o Δ zvýší a proto se všechny v síti rezerv objeví, pokud v ní již předtím nebyly. Podotýkám, že tyto opačné hrany nemusely být hranami v základní síti (to kupříklad platí pro většinu zpětných hran cesty v ukazovaných příkladech).

Scéna: *Řez a optimalita*

Předchozí scéna ukázala, jak Ford-Fulkersonův algoritmus zvyšuje tok pomocí zlepšujících cest. Na první pohled ovšem není jasné, zda v okamžiku, kdy algoritmus nenachází další zlepšující cestu, je nalezený tok největší možný. V této scéně ukážeme, že tomu tak je.

Pro zvolenou síť proveďte celý výpočet Ford-Fulkersonova algoritmu. Volba kroku je nastavena na nejvyšší hodnotu [**Celý výpočet**], takže je to možné provést jediným stisknutím knoflíku [**Krok**]. Můžete ale také volbu kroku změnit a výpočet provádět po větších či menších krocích postupně.

Řekneme, že vrchol sítě je dosažitelný, pokud do něho vede cesta složená z hran s kladnými rezervami. Zdroj je z triviálních důvodů dosažitelný, ale spotřebič v okamžiku výpočtu dosažitelný není, protože by se jinak Ford-Fulkersonův algoritmus nezastavil, ale upravoval by tok dále podle některé cesty, kterou lze spotřebič dosáhnout ze zdroje.

Provádějte výpočet až do jeho ukončení. V tento okamžik se barevné označení vrcholů změní. Všechny dosažitelné vrcholy se zbarví na stejnou barvu jako zdroj, tedy na zelenou. Všechny nedosažitelné vrcholy se zbarví na stejnou barvu jako spotřebič, tedy modrou. Dojde k rozdělení množiny vrcholů do dvou částí - dosažené a nedosažené, a toto rozdělení nazveme *řez*.

Změní se také barvy hran. Mějte na paměti, že nejsou zobrazeny všechny hrany, ale právě ty hrany, které mají kladnou rezervu. Z nich hrany spojující dva zelené dosažitelné vrcholy se také zbarví zeleně a hrany spojující dva modré nedosažitelné vrcholy se zbarví modře. Nakonec hrany vycházející z nedosažitelného vrcholu a končící ve vrcholu dosažitelném jsou obarveny červeně.

Povšimněte si, že nevidíme žádnou hranu vedoucí z nějakého dosažitelného vrcholu u do nějakého nedosažitelného vrcholu v, protože kdyby byla zobrazena, měla by kladnou rezervu a orientovaná cesta v síti rezerv ze zdroje do u (musí existovat, protože u je dosažitelný) spolu s hranou (u, v) by znamenala, že i v by byl dosažitelný, což by byl spor s naším výchozím předpokladem.

Nyní je myslím zcela jasné, že ze zelené části sítě se do modré části nedá žádným způsobem dopravit více toku, protože žádná z hran, které jsou v tomto směru, již nemá použitelnou rezervu. Matemeticky korektní důkaz tohoto tvrzení však podáme až ve škole.

Scéna: *Rychlost výpočtu*

V předchozí scéně jsme ukázali, že Ford-Fulkersonův algoritmus na rozdíl od hladového algoritmu vždy nalezne optimální řešení.

Jedinou otázkou je, jak dlouho to trvá. Myslím, že nebudete mít trpělivost dokončit výpočet, který bude probíhat na obrazovce, ale jeho myšlenku jistě pochopíte. Asi si všimnete, že Algovize cesty volí záměrně tak, aby výpočet nesměřoval k cíli příliš rychle. Mějte však na paměti, že Ford-Fulkersonův algoritmus nepředepisuje volbu cesty, pokud je více možností a proto je tento postup zcela legitimní. V případě irracionálních kapacit se dokonce může stát, že výpočet neskončí nikdy.

Zkuste si výpočet krokovat - uvidíte, že zpracování každé cesty zlepší tok o 1. Liché iterace naleznou cestu prochází ze zdroje přes horní vrchol do dolního vrcholu a pak do spotřebiče, což vede ke zvýšení toku v příčce na 1. Sudé iterace pak ze zdroje jdou nejprve do dolního vrcholu, protisměrně příčkou do horního vrcholu a pak do spotřebiče, což vede ke snížení toku příčkou na nulu a hra se může opakovat. Jelikož, jak jistě vidíte, je velikost maximálního toku v zobrazené síti 2000, trval by výpočet opravdu dlouho; jsem přesvědčen, že neprojdete ani prvních padesát cest, které Algovize nabídne.

Počítač by sice pro tuto síť výpočet dokončil v rozumné době, ale již v rámci 32-bitových celých čísel by bylo možno změnit kapacity hran na obvodu na zhruba 2 miliardy a tedy by počet iterací dosáhl okolo 4000000000 a použitím 64-bitových hodnot kapacit by se doba trvání výpočtu mohla zvýšit na hodnotu, přesahující současné (a zřejmě i budoucí) výpočetní možnosti lidstva. Je proto žádoucí nalézt rychlejší algoritmus, kde by především bylo možné stanovit horní odhad doby výpočtu jen ze znalosti počtu hran a vrcholů a nebylo by možno výpočet libovolně prodlužovat změnou kapacit. Dva takové algoritmy, Dinitzův a Goldbergův, budou popsány v další části.

4 Laboratoř

Síť je čtveřice $S = (G, c, z, s)$, kde $G = (V, H)$ je orientovaný graf, z, nazývaný *zdroj*, a s, nazývaný *spotřebič*, jsou dva různé vrcholy grafu G (tedy prvky množiny V) a c je funkce, která každé

hraně h grafu G (tedy každému prvku množiny H) přiřazuje nezáporné reálné číslo $c(h)$, nazývané *kapacita* hrany h. Kapacitu hrany $h = (u, v)$ budeme také označovat jako $c(u, v)$. Prvky množiny V budeme nazývat *vrcholy* sítě; vrcholy sítě jiné než zdroj a spotřebič se nazývají *vnitřní vrcholy* sítě. Prvky množiny H budeme nazývat *hrany* sítě.

Hranová funkce v síti S je funkce f, která každé hraně h sítě S přiřazuje číslo $f(h)$. Je-li $h = (u, v)$, pak mnohdy budeme místo $f(h)$ často psát $f(u, v)$. *Vrcholová funkce* je funkce g, která každému vrcholu v sítě přiřazuje číslo $g(v)$.

Pro množinu M vrcholů sítě označíme jako $in(M)$ množinu hran, které začínají mimo M a končí v M a jako $out(M)$ množinu hran, které začínají v M a končí mimo M. Pokud v je vrchol sítě, pak $in(v)$ respektive $out(v)$ jsou tyto hodnoty pro $M = \{v\}$, tedy $in(v)$ je množina hran vstupujících do v a $out(v)$ je množina hran, pro které je v jejich počátkem.

Přebytek hranové funkce t ve vrcholu u sítě je číslo

$$exc_{S,t}(u) = \sum_{h \in in(v)} t(h) - \sum_{k \in out(v)} t(k).$$

Zobecněný tok v síti S je hranová funkce t, pro kterou platí:

1. pro každou hranu h sítě je $0 < t(h) < c(h)$,
2. pro každý vrchol v sítě různý od zdroje je $exc_{S,t}(v) \geq 0$, a
3. jsou-li u a v dva vrcholy takové, že jak (u, v), tak i (v, u) jsou hrany sítě, pak buď $t(u, v) = 0$ a/nebo $t(v, u) = 0$.

Do definice zobecněného toku jsme tedy přidali požadavek nulové cirkulace, jak byl objasněn ve scéně o cirkulacích.

Tok v síti je speciální případ zobecněného toku t, pro který je $exc_{S,t}(u) = 0$ pro každý vnitřní vrchol sítě.

Je-li t zobecněný tok, pak přebytek spotřebiče, tedy číslo $exc_{S,t}(z)$, nazýváme *velikost* zobecněného toku t.

Z důvodů, které byly již ukázány v procházce algoritmem, budeme ve zbytku kapitoly předpokládat, že graf G je takový, že je-li (u, v) jeho hranou, pak i (v, u) je také jeho hranou, tedy s každou hranou v něm leží i hrana opačná. Pokud by se stalo, že v zadaném grafu tato podmínka splněna není, pak takové hrany do grafu přidáme s nulovou kapacitou. Tedy pokud $h = (u, v)$ by bylo hranou, ale (v, u) nikoli, pak do grafu přidáme hranu $h^{op} = (v, u)$ a položíme $c(h^{op}) = 0$. Byl-li t zobecněný tok v původní síti, pak jediná možnost jak jej dodefinovat v hraně h^{op} je položit $t(h^{op}) = 0$, protože musí být nezáporný, ale nejvýše rovný nulové kapacitě. V takovém případě se ale po přidání hrany a dodefinování t nezmění přebytky vrcholů, takže t zůstává zobecněným tokem nebo tokem a nezmění se ani jeho velikost. Hranu opačnou k hraně h budeme i v dalším vždy značit jako h^{op}.

Rezerva hrany $h = (u, v)$ sítě S vzhledem k zobecněnému toku t v této síti, označovaná jako $r_{S,t}(h)$, je číslo $c(h) - t(h) + t(h^{op})$. Nehrozí-li nedorozumění, budeme psát jen $r(h)$. Povšimněte si, že rezerva hrany může být větší než její kapacita, tedy speciálně rezerva může být kladná i když kapacita hrany je nulová; může se to ale stát jen když opačnou hranou h^{op} teče nenulový tok. Hrana h se nazývá nenasycená pokud její rezerva je kladná, v opačném případě se nazývá nasycená. Cesta v síti je posloupnost na sebe navazujících hran a nazývá se nenasycená, pokud je složena výhradně z nenasycených hran; pokud obsahuje alespoň jednu hranu nasycenou, budeme říkat, že je nasycená.

Zlepšující cesta (vzhledem k síti S a zobecněnému toku t) je posloupnost hran $h_1, \ldots, h_\ell$ sítě S taková, že rezervy všech těchto hran jsou kladné a existují vrcholy $v_0, \ldots, v_\ell$ takové, že $hi = (v_{i-1}, v_i)$ pro $i = 1, \ldots, \ell$.

Budeme velmi často používat následující operaci se zobecněným tokem t:

```
Podmínka aplikovatelnosti:  Δ ≤ min(c(h) − t(h) + t(h^op), exc_{S,t}(u))
Δ_1 ← min(Δ, t(h^op));
Δ_0 ← Δ − Δ_1;
t(h) ← t(h) + Δ_0;
t(h^op) ← t(h^op) − Δ_1;
```

Přesun přebytku Δ přes hranu $h = (u, v)$ (první varianta)

Podrobněji rozebráno tato operace pracuje následujícím způsobem:
je-li $t(h^{op}) > \Delta$, pak se $t(h^{op})$ sníží o Δ a $t(h)$ se nezmění,
je-li $\Delta > t(h^{op}) > 0$, pak se $t(h^{op})$ sníží na 0 a $t(h)$ se zvýší o $\Delta - t_0$, kde t_0 je původní hodnota $t(h^{op})$,
je-li $t(h^{op}) = 0$, pak se $t(h^{op})$ nezmění a $t(h)$ se zvýší o Δ.

Naše algoritmy ale budou v zájmu jednoduchosti pracovat nikoliv s toky, ale s rezervami. Jelikož budeme vycházet z nulového toku, jsou rezervy hran na začátku výpočtu rovny jejich kapacitám. V průběhu výpočtu budeme zaznamenávat, jak se rezervy hran mění, a teprve na konci si výše uvedeným způsobem z rezerv spočteme hodnoty toku, což jsou data, která nás zajímají. Tento postup má ale formální háček: rezerva hrany, tak jak byla definována, není proměnná, jejíž hodnotu je možno měnit, ale funkce, jejíž hodnota je dána velikostí toku. Proto budeme postupovat tak, že pro každou hranu h zavedeme proměnnou $R(h)$, jejíž hodnotu budeme měnit tak, abychom byli schopni dokázat, že $R(h)$ je stále rovna rezervě hrany h pro nějaký tok nebo zobecněný tok.

Obdobně jako s rezervami budeme zacházet i s přebytky. Přebytek vrcholu je funkce, kterou vypočítat může být náročné - je třeba uvážit toky všemi hranami, které do uvažovaného vrcholu vstupují nebo z něho vystupují, a těch může být mnoho. Zavedeme si proto také pro každý vrchol v proměnnou $E(v)$, která bude udržována tak, aby její hodnota se prokazatelně stále rovnala přebytku vrcholu vůči jistému toku nebo zobecněnému toku (který bude implicitně zaznamenán pomocí hodnot $R(h)$). V některých případech (jako právě zde u Ford-Fulkersonova algoritmu) není přebytky nutné znát a proto je možno proměnné E i příkazy s nimi manipulující vynechat, ale později u Goldbergova algoritmu bude jejich použití nutné a proto jsem je popsal již zde.

Pro úplnost shrneme, jak se proměnné R a E inicializují tak, aby odpovídaly rezervám a přebytkům vůči nulovému výchozímu toku:

```
for každou hranu h do R(h) = c(h);
for každý vrchol v do E(v) = 0;
```

Inicializace údajů o rezervách a přebytcích

Jestliže známe kapacitu jisté hrany h a její rezervu vůči nějakému toku, ale nevíme, kolik je tok touto hranou, pak není těžké si tuto hodnotu vypočíst (k tomu je nutný předpoklad nulové cirkulace). Pokud tedy budeme mít jistotu, že proměnné $R(h)$ jsou rovny rezervám hran vzhledem k jistému zobecněnému toku t, pak $t(h)$ je možno určit následujícím způsobem:

```
Vstup:      hrana h;
if c(h) > R(h)
then t(h) ← c(h) − r(h)
else t(h) ← 0;
```

Určení toku hranou ze znalosti její kapacity a rezervy

Naše algoritmy budou toky hranami a tedy i rezervy hran a přebytky vrcholů měnit výhradně aplikací výše uvedené operace přesunu přebytku. V řeči proměnných $R(h)$ a $E(v)$ tato operace vypadá takto:

```
Podmínka aplikovatelnosti:  Δ < min(R(h), E(u))
R(h) ← R(h) − Δ;
R(h^op) ← R(h^op) + Δ;
E(u) ← E(u) − Δ;
E(v) ← E(v) + Δ;
```

Přesun přebytku Δ přes hranu $h = (u, v)$ (druhá varianta)

Nyní již můžeme popsat Ford-Fulkersonův algoritmus:

```
Vstup:     Síť (G, z, s, c), kde G = (V, H)
Inicializace údajů o rezervách (a přebytcích);
loop:
sestroj zlepšující cestu;
if neexistuje zlepšující cesta then goto konec;
zpracuj zlepšující cestu;
gotostmt loop;
konec:  výpočet toků hranami z údajů o rezervách.
```

Ford-Fulkersonův algoritmus

Jednotlivé nedefinované části výpočtu jsou následující:

Sestrojení zlepšující cesty je nalezení cesty ze zdroje z do spotřebiče s v grafu, který má množinu vrcholů V a jako množinu hran má ty hrany $h \in H$, pro které platí $R(h) > 0$. Způsoby řešení této úlohy (například prohledávání do šířky nebo do hloubky) byly probrány v kapitole o prohledávání grafu a proto nejsou opakovány. V této části se také zjistí, zda taková cesta existuje, což je informace, která se použije v následujícím kroku algoritmu. Nalezená cesta je pak použita jako vstup pro zpracování zlepšující cesty.

Jak implicitně vyplývá z popisu algoritmu, uvažujeme v následujícím zpracování zlepšující cesty variantu přesunu přebytku hranou, která pracuje s proměnnými R (proměnné E zde nejsou potřeba):

```
Vstup:     Cesta h_1, ..., h_ℓ ze zdroje do spotřebiče
Proměnné:  číslo Δ
Δ ← min(R(h_1), ..., R(h_ℓ));
for i = 1, ..., ℓ do přesuň přebytek Δ přes hranu h_i;
```

Zpracuj zlepšující cestu

5 Škola

Nejprve jednoduché, ale často používané tvrzení o tom, že součet rezerv opačných hran zůstává konstantní:

Tvrzení 1 *[Labeled: FFSumRes] Je-li t zobecněný tok v síti $S = (G, z, s, c)$ a h její hrana, pak $r_{S,t}(h) + r_{S,t}(h^{op}) = c(h) + c(h^{op})$.*

Důkaz: Jelikož $(h^{op})^{op} = h$, platí $r_{S,t}(h) + r_{S,t}(h^{op}) = (c(h) - t(h) + t(h^{op})) + (c(h^{op}) - t(h^{op}) + t(h)) = c(h) + c(h^{op})$. ♣

Pojem přebytku vrcholu si nyní zobecníme na přebytek množiny vrcholů:

Definice 1 *[Labeled: FFDefExc] Nechf je dána síf $S = ((V, H), z, s, c)$, zobecněný tok t v síti S a množina M vrcholů sítě. Pak*

$$exc_{S,t}(M) = \sum_{h \in (M)} t(h) - \sum_{k \in out(M)} t(k).$$

Důležité pomocné lemma, které bude nyní dokázáno, je tvrzení o přebytku množiny vrcholů:

Tvrzení 2 *[Labeled: FFExcM] Nechť t je hranová funkce v síti S, M je množina vrcholů této sítě. Pak*

$$exc_{S,t}(M) = \sum_{u \in M} exc_{S,t}(u).$$

Důkaz: Označme

$$X1 = \sum_{u \in V} \sum_{h \in in(Vu)} t(h), \quad X2 = \sum_{u \in V} \sum_{h \in out(u)} t(h),$$

$$Y1 = \sum_{h \in in(M)} t(h), \quad Y2 = \sum_{h \in out(M)} t(h).$$

Pak je zřejmé, že pravá strana rovnosti z lemmatu, součet přebytků vrcholů z množiny M vzhledem k t, je rovna $X1 - X2$ a levá strana je $Y1 - Y2$.

Pro každou hranu $h = (v, w)$ takovou, že $v, w \in M$ se $t(h)$ objeví právě jednou jako sčítanec v $X1$ (jako prvek $in(u)$ pro $u = w$) a právě jednou jako sčítanec v $X2$ jako prvek $out(u)$ pro $u = v$), tedy se ve výrazu $X1 - X2$ vyruší. Jelikož h nepatří ani do $in(M)$, ani do $out(M)$, $t(h)$ se neobjeví jako sčítanec ani v $Y1$ ani v $Y2$.

Pro každou hranu $h = (v, w)$ takovou, že $v, w \notin M$ se $t(h)$ neobjeví jako sčítanec v žádném z výrazů $X1$, $X2$, $Y1$, $Y2$.

Pro každou hranu $h = (v, w)$ takovou, že $v \notin M$ a $w \in M$ se $t(h)$ objeví právě jednou jako sčítanec v $X1$ (jako prvek $in(u)$ pro $u = w$), ale ne jako sčítanec v $X2$, tedy k výrazu $X1 - X2$ přispívá hodnotou $t(h)$. Jelikož h patří do $in(M)$, ale nepatří do $out(M)$, hrana h k výrazu $Y1 - Y2$ přispívá také hodnotou $t(h)$.

Nakonec pro každou hranu $h = (v, w)$ takovou, že $v \in M$ a $w \notin M$ se $t(h)$ objeví právě jednou jako sčítanec v $X2$ (jako prvek $out(u)$ pro $u = v$), ale ne jako sčítanec v $X1$, tedy k výrazu $X1 - X2$ přispívá hodnotou $-t(h)$. Jelikož h nepatří do $in(M$, ale patří do $out(M$ hrana h k výrazu $Y1 - Y2$ přispívá také hodnotou $-t(h)$.

Jelikož jiné členy se ve výrazu neobjevují, je zřejmé, že $X1 - X2 = Y - Y2$. ♣

Jako okamžité důsledky dostáváme

Tvrzení 3 *[Labeled: FFSumExc] Součet přebytků všech vrcholů vzhledem k libovolné hranové funkci t je 0.*

Důkaz: Viz minulé lemma pro M rovné množině všech vrcholů (kdy $in(M)$ a $out(M)$ jsou prázdné množiny). ♣

Tvrzení 4 *[Labeled: FFFlowSize] Je-li t tok, pak $ext_{S,t}(z) = -ext_{S,t}(s)$, kde z a s jsou zdroj a spotřebič sítě.*

Důkaz: Viz minulé lemma, uvážíme-li že přebytky vnitřních vrcholů sítě vzhledem k toku jsou 0. ♣

Nyní si dokážeme tvrzení, které říká, že proměnné R a E zavedené v laboratoři jsou skutečně rovny hodnotám rezerv a přebytků:

Tvrzení 5 *[Labeled: FFR] Nechť je dána síť $S = ((V, H), z, s, c)$, zobecněný tok t v síti S a proměnné $R(h)$, $h \in H$, a $E(v)$, $v \in V$ pro něž platí, že*

$$R(h) = r_{S,t}(h) \text{ pro každou hranu } h \qquad (A)$$

$$E(v) = exc_{S,t}(v) \text{ pro každý vrchol } v \qquad B)$$

Nechť $h = (u, v)$ je hrana sítě S. Pak pokud se hodnoty $t(h)$ a popřípadě $t(h^{op})$ změní první variantou přesunu přebytku Δ přes hranu h a hodnoty $R(h)$ a $R(h^{op})$ i hodnoty $E(u)$ a $E(v)$ se změní druhou variantou přesunu přebytku Δ přes hranu h, platnost podmínek (A) i (B) zůstane zachována; přitom první varianta přesunu přebytku Δ je aplikovatelná právě tehdy, když je aplikovatelná druhá varianta pro stejnou hodnotu Δ.

Důkaz: Tvrzení o aplikovatelnosti plyne přímo z podmínek (A) a (B) před provedením přesunu. První varianta přesunu zvýší $t(h)$ o Δ_0 a sníží $t(h^{op})$ o Δ_1, takže $r_{S,t}(h) = c(h) - t(h) + t(h^{op})$ poklesne o $\Delta_0 + \Delta_1 = \Delta$ (označení viz popis první varianty přenosu přebytku) a o totéž druhá varianta přesunu sníží $R(h$.

Jelikož součet reserv hran h a h^{op} je podle Lemmatu 1 konstantní, dojde současně při první variantě přesunu ke zvýšení rezervy hrany h^{op} o Δ a stejná změna se provede s $R(h^{op})$.

Přesun může změnit tok jen v hranách h a h^{op}, toky ostatními hranami se nemění. Proto se mohou změnit jen přebytky dvou vrcholů u a v. Změna přebytku vrcholu u je rovna změně výrazu $t(h) + t(h^{op})$, protože h z vrcholu u vystupuje a h^{op} do něj vstupuje, a již jsme ukázali, že tento výraz poklesne o Δ; stejně tak se změní i $E(u)$.

Nakonec jelikož součet přebytků je podle Lemmatu 3 konstantní, přebytek vrcholu v o Δ vzroste a stejně se změní i $E(v)$. ♣

Nyní dokážeme, že převádění přebytku nejvýše rovného rezervě hrany nikdy neporuší první a třetí podmínku z definice toku; ze zobecněného toku se tedy libovolným převedením stane zase zobecněný tok. Tento poznatek bude použit i u Dinitzova a Goldbergova algoritmu.

Tvrzení 6 *[Labeled: FFTransfCorrect] Nechť t je zobecněný tok, h je hrana, Δ je číslo takové, že $0 < \Delta < r_t(h)$. Pak převedení přebytku velikosti Δ hranou h neporuší kapacitní omezení $0 < t(k) < c(k)$ ani podmínku $t(k) \cdot t(k^{op}) = 0$ pro žádnou hranu k sítě.*

Důkaz: Převodem přebytku přes hranu h se může změnit tok jen hranami h a h^{op}. Uvažte nyní, že kapacity hran se během výpočtu nemění. Označme τ a τ^{op} toky hranami h a h^{op} před převedením. Rezerva hrany h před převedením tedy byla $c(h) - \tau + \tau^{op}$.

Jelikož předpokládáme, že Δ je nejvýše rovno rezervě hrany h před převedením, platí

$$\tau + \Delta - \tau^{op} \leq \tau + (c(h) - \tau + \tau^{op}) - \tau^{op} = c(h).$$

Úvahu rozdělíme na tři případy:
pokud je $\tau^{op} > \Delta$, pak se $t(h^{op})$ sníží a to na hodnotu $\tau^{op} - \Delta > 0$ protože $\Delta < c(h) - \tau + \tau^{op} < \tau^{op}$; kapacitní omezení pro h^{op} tudíž zůstane v platnosti; $t(h)$ se nezmění;
pokud je $\Delta > \tau^{op} > 0$, bude po převedení $t(h^{op}) = 0$, tedy vyhovující, a $t(h)$ se zvýší, tedy bude nezáporné a bude mít hodnotu $\tau + \Delta - \tau^{op} < c(h)$, tedy (jak bylo dokázáno výše) nejvýše $c(h)$;
pokud je $\tau^{op} = 0$, pak se tok hranou h^{op} nezmění, (h) se zvýší, tedy bude nezáporný a bude mít hodnotu $\tau + \Delta = \tau + \Delta - \tau^{op}$, tedy nejvýše $c(h)$.

Důkaz tvrzení týkající se nulovosti jednoho z $t(h)$ a $t(h^{op})$ rozdělíme na tři případy:
pokud před převedením je $t(h^{op}) > \Delta$, pak se $t(h)$ se nezmění a zůstane nulový;
pokud před převedením je $\Delta > t(h^{op}) > 0$, bude po převedení $t(h^{op})$ nulový;
pokud před převedením je $t(h^{op}) = 0$, pak se převedením nezmění a zůstane nulový. ♣

Dále dokážeme, že

Tvrzení 7 *[Labeled: FFPathCorrect] Zpracování cesty podle Ford-Fulkersonova algoritmu nezmění přebytek žádného vnitřního vrcholu sítě a velikost toku zvýší o Δ.*

Důkaz: Na základě Lemmatu 5 můžeme místo přebytků uvažovat proměnné $E(v)$ pro vrcholy v sítě.

Pro vrcholy mimo cestu se zjevně hodnoty E nemění.

Je-li v vrchol zpracovávané cesty jiný než její počátek (zdroj) nebo konec (spotřebič), pak je koncem právě jedné z hran cesty a začátkem právě jedné z hran cesty. Při zpracování cesty se tedy $E(v)$ zvýší o Δ při přenosu hranou vstupující do v, ale zase sníží při přenosu hranou vystupující z v, takže má po ukončení zpracování cesty stejnou hodnotu jako na začátku.

Do spotřebiče vede jediná z hran cesty a žádná z něho nevystupuje, proto se přebytek spotřebiče zvýší o Δ. ♣

Celkově jsme tedy dokázali, že při každém vstupu a výstupu do těla smyčky ve Ford-Fulkersonově algoritmu je t tok a jeho velikost stále roste. Je ovšem třeba ukázat, že se algoritmus vždy zastaví a nalezne největší možný tok.

Se zastavením je potíž. Jsou známy sítě s irracionálními kapacitami, pro něž se Ford-Fulkersonův algoritmus při nevhodných volbách cest nikdy nezastaví. Jsou-li všechny kapacity hran celočíselné, pak i každá jiná hodnota (tok hranou, rezerva, hodnota Δ pro zpracování cesty) je také celočíselná a proto Δ, což je kladné celé číslo, bude alespoň 1 a proto se tok po každém zpracování cesty zvýší alespoň o 1 a tedy se po konečné době výpočet zastaví.

Jsou-li kapacity hran racionální, pak napište kapacity jako zlomky s celočíselným čitatelem a jmenovatelem a vynásobte je nejmenším společným násobkem jmenovatelů. Získá se síť s celočíselnými kapacitami, ve které výpočet probíhá stejně jako v původní síti, takže se po jisté době také zastaví. Tato doba může ovšem být dlouhá, jak jsme viděli během procházky.

Nyní ukážeme, že pokud se Ford-Fulkersonův algoritmus zastaví, dá největší možný tok. Nejprve zavedeme následující pojem

Definice 2 *[Labeled: FFCut] Řez je libovolná množina M vrcholů, která obsahuje zdroj a neobsahuje spotřebič. Velikost řezu M je součet kapacit hran patřících do $out(M)$.*

Nyní jedno technické lemma

Tvrzení 8 *[Labeled: FFCutFlow] Nechť t je tok a M je řez v síti S. Pak velikost toku t je rovna $-exc_{S,t}(M)$.*

Důkaz: Podle definice velikosti toku a Tvrzení 3 je velikost toku rovna hodnotě $-exc_{S,t}(z)$, kde z je zdroj. V množině M kromě zdroje jsou již jen vnitřní vrcholy sítě, které mají přebytek nulový a tedy podle Tvrzení 2 je přebytek množiny M roven hodnotě $exc_{S,t}(z)$. ♣

Jedno z nejdůležitějších tvrzení teorie toků v sítích je

Věta 9 *[Labeled: FFWeakMinMax] Nechť t je tok a M je řez v síti. Pak velikost toku t je menší nebo rovná velikosti řezu M.*

Důkaz: Velikost toku t je podle výše uvedeného lemmatu rovna

$$-exc_{S,t}(M) = \sum_{h \in out(M)} t(h) - \sum_{k \in (M)} t(k) \leq \sum_{h \in out(M)} t(h),$$

kde jsme využili toho, že pro hrany $h \in in(M)$ je $t(h) \geq 0$ a pro hrany $h \in out(M)$ je $t(h) < c(h)$. Hodnota na pravé straně nerovnosti je ale velikost řezu M. ♣

Je zřejmé, že pokud se Ford-Fulkersonův algoritmus zastaví, pak neexistuje cesta ze zdroje do spotřebiče, která je složena z hran, které mají kladnou rezervu vzhledem k výslednému toku, protože jinak by některá z takových cest byla vzápětí použita pro zvýšení toku. Optimalita výsledku určeného Ford-Fulkersonovým algoritmem proto plyne z následující věty

Věta 10 *[Labeled: FFMinMax] Nechť t je tok takový, že každá cesta ze zdroje do spotřebiče je nasycená. Pak neexistuje tok, který by měl větší velikost než t.*

Důkaz: K důkazu věty stačí dokázat, že existuje řez M, který má velikost stejnou jako tok t, protože podle předchozí věty neexistuje tok, který by měl větší velikost než je velikost řezu M. Definujme M jako množinu všech vrcholů, do kterých ze zdroje vede nenasycená cesta. Zdroj do množiny triviálně patří, z podmínky ve větě plyne, že do ní nepatří spotřebič a proto je M řez.

Nechť $h = (v, w)$ je hrana patřící do $out(M)$. Jelikož tedy $v \in M$, pak podle definice M existuje nenasycená cesta ze zdroje do v. Navíc $w \notin M$ a tedy musí být $t(h) = c(h)$, protože jinak by h byla nenasycená a tedy nenasycená cesta ze zdroje do v doplněná o hranu h by dávala nenasycenou cestu ze zdroje do w ve sporu s definicí M.

Nechť nyní $h = (v, w)$ je hrana patřící do $in(M)$. Jelikož tedy $w \in M$, pak podle definice M existuje nenasycená cesta ze zdroje do w. Navíc $v \notin M$ a tedy musí být $t(h) = 0$, protože jinak by h^{op} měla kladnou rezervu, a tedy nenasycená cesta ze zdroje do v doplněná o nenasycenou hranu h^{op} by byla nenasycená cesta ze zdroje do v ve sporu s definicí M.

Podle Tvrzení 8 je velikost toku t rovna

$$\sum_{h \in out(M)} t(h) - \sum_{k \in (M)} t(k) \leq \sum_{h \in out(M)} t(h)$$

a výraz na pravé straně rovnosti je velikost řezu M. ♣

Jako okamžitý důsledek tedy dostáváme

Věta 11 *Pokud se Ford-Fulkersonův algoritmus zastaví, nalezne největší možný tok.*

6 Dinitzův algoritmus

Scéna: *Dinitzův algoritmus*

Dinitzův algoritmus je vlastně implementací Ford-Fulkersonova algoritmu. Ford-Fulkersonův algoritmus nepředepisuje, jaká zlepšující cesta (t.j. nenasycená cesta ze zdroje do spotřebiče) se má vybrat, pokud je jich k dispozici více. Dinitzův algoritmus stanoví, že je třeba vybrat tu z nich, která obsahuje nejméně hran. Jak uvidíme, tento jednoduchý trik způsobí překvapivou změnu nejhoršího možného chování algoritmu. Nezávisle jej objevili také Edmonds a Karp, ale Dinitzova implementace navíc obsahuje v každé své fázi předzpracování, které výpočet ještě urychlí.

Základní schéma algoritmu je zobrazeno na displeji. Provádějte výpočet krokovacími knoflíky; jednotlivé fáze výpočtu zde budou komentovány:

Krok: *Inicializace*

Zde se provedou jednoduché úvodní operace, především určení počátečních rezerv hran, které nebudu blíže komentovat.

Krok: *Určení vrstev*

Vrcholy rozdělíme do vrstev; vrchol v je v k=té vrstvě, pokud nejkratší cesta ze zdroje do spotřebiče, tvořená hranami s kladnou rezervou, obsahuje k hran. V nulté vrstvě je tedy jen zdroj, v první vrstvě vrcholy, do kterých vede ze zdroje cesta s kladnou rezervou, v další vrstvě vrcholy, do kterých vedou hrany z první vrstvy atd. Vrstva jsou naznačeny svislými pásy.

Obecně se může stát, že do některého vrcholu v síti rezerv cesta ze zdroje vůbec nevede. Takové vrcholy by byly zobrazeny u pravého okraje obrazovky.

Když se vrcholy dělí do vrstev poprvé, jsou toky hranami rovny nule a tedy jejich rezervy jsou rovny kapacitám, ale rozdělování do vrstev se bude provádět opakovaně a později již bude vazba mezi rezervami a kapacitami minimální a rozdělení do vrstev může být zcela odlišné.

Krok: *Test ukončení výpočtu*

Pokud by v síti nevedla ze zdroje do spotřebiče cesta tvořená nenasycenými hranami, což bychom při určování vrstev zjistili, pak výpočet končí vyskočením z vnějšího cyklu a provedením závěrečných úprav; jak jsme již ukázali dříve, znamená to, že již byl nalezen maximální tok.

Krok: *Vynechání pomalých hran*

Do tohoto kroku se dostaneme, pokud při určování vrstev bylo mimo jiné zjištěno, že v síti existuje nenasycená cesta ze zdroje z do spotřebiče s. $L(s)$ pak udává délku této cesty. Dinitzův algoritmus přikazuje, aby pro zlepšování toku byla v síti rezerv vybrána ta cesta ze zdroje do spotřebiče, které má minimální délku, tedy délku rovnou $L(s)$.

Pozorováním výpočtu se zjistí, že většinou takových cest bývá více, někdy i velké množství a řadu z nich postupně vybereme pro zlepšení toku. Proto je užitečné okamžitou situaci prozkoumat podrobněji a provést předběžné výpočty, díky nimž bude snazší tyto cesty hledat. Toto předzpracování, které schází v Edmonds-Karpově algoritmu, způsobí, že Dinitzův algoritmus má lepší asymptotický odhad nejhoršího možného chování.

Podíváme-li se na graf na obrazovce, je zřejmé, že nasycená cesta ze zdroje do spotřebiče (tedy cesta skládající se z viditelných hran) a obsahující minimální počet hran se nemůže nikde zdržovat a každou hranou musí postoupit o jednu vrstvu doprava. Takové hrany budeme nazývat *rychlé*. Hraně, která má oba konce ve stejné vrstvě a nebo dokonce vede směrem vlevo (její konec je zdroji blíže než její počátek), které budeme říkat *pomalá*. Takovou hranou nejkratší nenasycená cesta procházet nemůže.

Stejně tak nemůže cesta minimální délky procházet vrcholy, které mají vzdálenost od zdroje větší nebo rovnou vzdálenosti spotřebiče od zdroje (a jsou různé od spotřebiče). Hrany vycházející z takových vrcholů také označíme za pomalé.

Pomalé hrany se poznají snadno na základě hodnot L. Má-li hrana počátek v a konec w, pak je pomalá právě když $L(w) \leq L(v)$ nebo $L(v) \geq L(s)$, kde s je spotřebič.

Na obrazovce se v tomto kroku pomalé hrany označí žlutě, rychlé hrany zůstávají modré.

Žádnou pomalou hranou (nyní je žlutá), nemůže procházet nejkratší nenasycená cesta ze zdroje do spotřebiče. Naopak to ale zjevně neplatí: leckterá modrá hrana je taková, že přes ni také taková cesta nevede. Takových hran se ale zbavíme později.

Tento krok je natolik jednduchý, že jej již podrobněji rozebírat nebudu.

Zatržení checkboxu [**Skryj žluté**] (možná musíte nejprve stisknout knoflík [**Více ovladačů**]) se žluté pomalé hrany přestanou zobrazovat vůbec. Zobrazování výpočtu je pak názornější, ale zakrývá fakt, že v síti mají kladnou rezervu i pomalé hrany.

Krok: *Nalezení slepých konců*

Jestliže z vrcholu vycházejí jen pomalé hrany, pak přes něj určitě nevede nejkratší nenasycená cesta. Totéž platí i pokud do vrcholu vstupují jen pomalé hrany. Takové vrcholy nyní určíme. Tato jednoduchá operace bude později rozebrána podrobněji, ale jistě by vám nečinilo problém příslušný podprogram napsat sami.

Krok: *Pročištění sítě*

Jak jsme již viděli v předchozím kroku, přes některé rychlé (modré) hrany nevede žádná nejkratší nenasycená cesta ze zdroje do spotřebiče, protože pokud přes ně přejdeme, dostaneme se dříve či později do slepé uličky, neboli do vrcholu (jiného než spotřebič), ze kterého žádné rychlé modré hrany nevystupují.

Podobně je tomu i u hran, do kterých se ze zdroje nemůžeme dostat rychlými hranami.

V tomto kroku se všechny slepé uličky odstraní - hrany, které jsou sice rychlé, ale vedou do slepé uličky nebo se do nich dostaneme jen ze slepé uličky a ne ze zdroje. Jak se to provede ukážeme později, v této scéně uvidíme jen výsledek a připomeneme, že v některých případech může být slepá ulička dlouhá a až za dlouhou dobu se ukáže, že nikam nevede.

Podobně jako pomalé hrany se slepé rychlé hrany po provedení kroku buď zobrazují jako žluté nebo nezobrazují v závislosti na nastavení checkboxu [**Skryj žluté**].

Zkuste se vrátit k síti před pročištěním a pak si výsledek operace znovu prohlédout; je zřejmé, že po pročištění leží *každá* modrá hrana na některé nejkratší nenasycené cestě ze zdroje do spotřebiče a ji možno snadno nalézt bez vracení se a navíc *každá* cesta ze zdroje do spotřebiče tvořená modrými hranami je nenasycená cesta obsahující minimální počet hran.

Z důvodů, které poznáme později, je následující část algoritmu souhrnně označena jako nalezení nasyceného toku modrými hranami. Jedná se ale, jak uvidíte při krokování o cykl sestávající z následujících kroků:

Krok: *Test ukončení cyklu hledání nalezení nasyceného toku modrými hranami*

Během zlepšování toku podél nalézených cest a následného dočišťování - viz následující dva kroky - ubývá modrých hran. Jakmile po zpracování některé cesty zmizí poslední modrá cesta ze zdroje do spotřebiče, pak se všechny modré hrany stanou slepými a následující dočištění je všechny vynechá. Pak budeme muset další cestu hledat mezi hranami, které zatím byly ohodnoceny jako pomalé. Proto se probíraný cykl ukončí a vracíme se znovu na rozdělení vrcholů do vrstev, které některé hrany dosud označené jako pomalé přehodnotí na rychlé a výpočet obecně může pokračovat.

Krok: *Nalezení a zpracování cesty*

Hledání nejkratší nenasycené cesty bude probráno později, v této scéně se cesta zobrazí okamžitě.

Způsobem, který známe z Ford-Fulkersonova algoritmu nyní nalezneme minimální rezervu na cestě a o tuto hodnotu se sníží rezervy hran na cestě a naopak zvýší rezervy opačných hran. Množina nenasycených hran se tedy změní dvojím způsobem

- pro alespoň jednu hranu nalezené cesty se její rezerva *sníží na nulu* a tedy tato hrana přestane být zobrazována a algoritmem (alespoň dočasně) uvažována; je to hrana nebo hrany s minimální rezervou mezi hranami cesty;
- hranám, které leží v protisměru k hranám cesty, rezerva vzroste; po provedení operace tedy tyto hrany jsou mají kladnou rezervu; některé z nich ji měly kladnou již předtím, ale i pokud byla rezerva nulová, zpracováním cesty vzrostla na kladnou hodnotu.

Na úplné provedení tohoto kroku musíme zmačknout knoflík [**Krok**] čtyřikrát.

Po prvním klepnutí na knoflík se objeví zeleně vyznačená cesta. Po druhém se určí a zobrazí hodnota Δ minimální rezervy hran cesty. Po třetím klepnutí se barevně zobrazí změny v síti rezerv:

- hrany, kterým rezerva vzrostla, jsou hrany opačně orientované k hranám cesty a zobrazí se ve dvou odstínech červené; hrany, které již předtím byly nenasycené, budou světle červené, zatímco hrany, jejíchž rezerva vzrostla z nuly na kladné číslo a které tedy se nenasycenými právě staly, budou sytě červené;
- hrany cesty, kterým rezerva poklesla, ale nikoli na nulu, takže zůstávají nenasycené, zůstanou zelené a
- hrany cesty, kterým rezerva poklesla na nulu, takže přestanou být nenasycené, budou nyní bílé a neobtažené.

U hran budou také uvedeny změněné rezervy.

Nakonec po čtvrtém klepnutí se barevné označení hran vrátí k obvyklému způsobu: hrany cesty, které přestaly být nenasycené, nejsou zobrazeny, ostatní hrany cesty jsou zobrazeny modře.

Opačně orientované hrany, které v minulém kroku byly červené, *jsou pomalé* a proto budou zobrazeny žlutě nebo neznázorněny v závislosti na nastavení checkboxu [**Skryj žluté**]. Tento fakt je klíčový pro pochopení algoritmu; zpracováním cesty sice některé hrany nenasycené přestávají být, ale jiné se nenasycenými mohou stát, ale nové nenasycené hrany *nejsou* použitelné pro vytvoření nejkratší nenasycené cesty.

Jak bylo řečeno výše, alespoň jedna hrana cesty přestane být nenasycená. Tím se opět mohou vytvořit slepé uličky. Dinitzův algoritmus proto po každém zpracování cesty znovu volá pročištění sítě, které bylo zmíněno výše.

Krok: *Dočištění*

Po zpracování cesty vypadla alespoň jedna modře zobrazená hrana. To může vést k tomu, že některé modré rychlé hrany se stanou slepými - nevede přes ně žádná modrá cesta sdo spotřebiče. Takové hrany pak v tomto kroku vynecháme stejně tak, jako se to provádělo v kroku čištění uvedeném výše.

Scéna: *Proč krátké cesty*

Projděte si výpočet ještě jednou; krokování je ještě hrubší než v předchozí scéně, protože u zpracování cesty je znázorněn jen výsledek po jejím ukončení. Výpočet si rozdělíme na *fáze*, fáze jsou navzájem odděleny skokem na návěští "loop" a přepočítání vrstev.

Po celou dobu trvání fáze má nejkratší cesta ze zdroje do spotřebiče v síti rezerv stejnou délku, danou počtem vrstev, označme ji L. Jak jsme viděli už v předchozí scéně, počet cest této délky L každým zpracováním *klesne* alespoň o jednu (vypadne přinejmenším zpracovávaná cesta: alespoň jedna hrana zpracovávané cesty se nasytí) a mohou sice vzniknout nové nenasycené cesty ze zdroje do spotřebiče, ale žádná nová modrá rychlá modrá *nepřibude* a tedy nepřibude ani žádná modrá cesta *stejné nebo kratší délky*.

Dokud existuje alespoň jedna nenasycená cesta délky L, dočištění ponechává síť modrých hran neprázdnou (hrany cesty pročištění nemůže vyhodit). Jakmile ale všechny cesty této délky zaniknou, dočištění všechny zbývající modré hrany vyhodí a proto vzápětí nastává výskok z vnitřního cyklu a nová fáze. V tomto okamžiku se také zjevně zvýší délka nejkratší modré cesty ze zdroje do spotřebiče (původní cesty délky L zanikly a kratší cesty nebo jiné cesty stejné délky v síti nebyly a ani nevznikly). Po ukončení libovolné fáze se proto délka nejkratší modré cesty ze zdroje do spotřebiče *zvětší*.

Zkuste si nyní krokovat výpočet znovu a tvrzení si ověřit. V rohu obrazovky se zobrazuje pořadové číslo fáze a délka nejkratší cesty.

Z uvedeného ihned plyne, že počet fází, neboli počet opakování vnějšího cyklu algoritmu nemůže být větší než $n-1$, kde n je počet vrcholů sítě. Prostá cesta ze zdroje do spotřebiče totiž nemůže mít více než $n-1$ hran, ale alespoň jednu hranu mít musí. Nyní stačí uvážit, že délka nejkratší cesty ze zdroje do spotřebiče může mít jen hodnoty $1, \ldots, n-1$, ale v každé fázi jinou.

Kromě toho při každém provedení těla vnitřního cyklu zmizí alespoň jedna modrá hrana, takže počet provedení vnitřního cyklu v rámci jedné fáze nepřevýší počet hran, které byly modré při vstupu do cyklu (a tedy je nejvýše roven počtu všech hran sítě).

Scéna: *Proč ne dlouhé cesty*

Tato scéna je shodná s předchozími scénami ukazujícími činnost algoritmu s jedinou, ale podstatnou odchylkou: způsob výběru cesty pro zlepšení toku nepreferuje nejkratší nenasycenou cestu, ale naopak cestu, která se snaží postupovat co nejpomalejším způsobem. Jde vlastně o Ford-Fulkersonův algoritmu, u kterého znázorňujeme rozdělení vrcholů do vrstev.

Na cestě se nyní může objevit i pomalá hrana. V případě výchozí sítě této scény se do cesty dostane hrana vedoucí o dvě vrstvy zpět ke spotřebiči.

Jestliže některá hrana h cesty vede o alespoň dvě vrstvy zpět, hrana h^{op} k ní opačně orientovaná vede o alespoň dvě vrstvy dopředu směrem ke spotřebiči. Znamená to, že h^{op}, která je po zpracování cesty určitě nenasycená, se stane "superrychlou" nenasycenou hrana, která umožní od zdroje do spotřebiče postupovat rychleji, než tomu bylo dosud možné.

Pokud cesta používá pomalé hrany, vedoucí o jednu vrstvu zpět, nevytvoří to sice kratší nenasycenou cestu, ale může to vést ke vzniku nové a stejně dlouhé nenasycené cesty.

Vidíme, že výběr dlouhých zlepšujících cest by vedl k tomu, že zpracování jedné cesty jisté délky by mohlo vést k porušení tvrzení, že nevznikají cesty, složené z hran s kladnou rezervou, které by byly stejně dlouhé nebo kratší než existující cesty. Přitom na tomto předpokladu je podstatným způsobem závislý odhad počtu opakování cyklů kódu, který byl uveden v minulé scéně.

Scéna: *Vrstvy*

Tato scéna umožňuje podrobné krokování etapy určování vrstev sítě rezerv. Podrobnější ji ale komentovat nebudeme, protože se jedná o prohledávání sítě do šířky, které bylo podrobně probráno již dříve.

Scéna: *Čištění*

Pročištění sítě je sice rutinní, ale ne zcela jednoduchá operace. Výhodné je provádět ji jak je popsáno dále a jak je možno podrobně krokovat na obrazovce, zatímco ostatní části výpočtu se krokují velice hrubě.

Předně si pro každý vrchol poznamenáváme, kolik modrých rychlých hran z něho vychází a kolik modrých rychlých hran do něho vstupuje. Potom vedeme záznam o vrcholech, ze kterých žádná modrá hrana nevystupuje, ale některé vstupují a záznam o vrcholech s opačnou vlastností (žádná modrá hrana nevstupuje, některé vystupují).

Po rozdělení vrcholů do vrstev a stanovení rychlých modrých hran a žlutých pomalých hran se tyto údaje spočítají zřejmým byť poněkud pomalým způsobem, pak se již jen upravují.

Pak se po sobě zpracují slepé uličky typu nic nevychází a typu nic nevstupuje. Popíšeme zde první operaci, druhá je obdobná.

Do množiny Q vložíme všechny vrcholy, do kterých vstupuje alespoň jedna modrá hrana, ale žádná modrá hrana z něj nevystupuje. Pak dokud je Q neprázdná, opakujeme následující operaci:

z Q se vyjme libovolný vrchol v a pro všechny hrany (u, v) vstupující do v se provede následující:

hrana se přebarví z modré na žlutou, o 1 se sníží údaj o počtu modrých hran vystupujících z u a pokud toto číslo poklesne na 0, vrchol u se zařadí do množiny Q.

Krokujte si výpočet a sledujte, jak se slepé uličky zpracovávají.

Bystrý čtenář si jistě povšiml, že konstrukce a zpracování fronty vrcholů do kterých nevstupují modré hrany je zbytečné: slepé uličky je nutno odstranit, abychom se do nich nedostali při hledání cesty do spotřebiče v čisté síti, vycházejíce přitom ze zdroje, protože by bylo nutno couvat a výpočet by se zpomalil. Vrcholy, do kterých nevstupují modré hrany, jsou ale vrcholy, do nichž se při takovém hledání nedostaneme; proto při hledání cesty nemohou působit komplikace a není tudíž potřeba ztrácet čas jejich odstraňováním. V appletu jsou zařazeny proto, aby nákres sítě modrých hran byl přehlednější a "čistší". Jelikož zpracování obou front je analogické, nepředstavuje jistě pro čtenáře žádnou zátěž při čtení knihy.

Scéna: *Zpracování cesty*

Tato scéna jen opakuje obdobnou scénu z Ford-Fulkersonova algoritmu. Zařadili jsme ji jen proto, abyste si dobře povšimli, že při hledání cesty v čisté siti se nemůže stát, že bychom zabloudili do slepé uličky, museli se vracet a tím by se hledání prodloužilo. V čisté síti s každým krokem posuneme o jednu vrstvu blíže ke spotřebiči a proto (na rozdíl od Ford-Fulkersonova algoritmu, ale i od algoritmu Edmondse a Karpa) doba potřebná k nalezení cesty je úměrná její délce.

7 Laboratoř

Ve výkladu Dinitzova algoritmu používáme všechna označení, zavedená v sekci o Ford-Fulkersonově algoritmu, jehož je Dinitzův algoritmus speciálním případem. Nejprve si zavedeme označování stavu hran. Pro každou hranu h bude k dispozici proměnná $h.status$, která bude nabývat 4 možných hodnot - nasycená, pomalá, slepá, použitelná (a po jistou dobu na začátku výpočtu může být proměnná neinicializovaná). Nasycené hrany budou ty, které se při procházce algoritmem neznázorňovaly, použitelné hrany byly označovány modře, pomalé a nepoužitelné hrany dvěma odstíny žluté. Hrana, která je bud' ve stavu "slepá" nebo "použitelná" se také někdy označuje jako rychlá a hrana, která není ve stavu "nasycená" (je tedy pomalá, slepá nebo použitelná) bude nazývána nenasycená.

Dále budeme pro každý vrchol u používat celočíselné proměnné $L(u)$, $D_{in}(u)$ a $D_{out}(u)$.

Dinitzův algoritmus může být popsán následujícím způsobem:

```
 1 DAinit     inicializace údajů o rezervách;
 2 DAloop    Fáze:
 3 DAlaye        určení vrstev;
 4 DAtest        if neexistuje nenasycená cesta zdroj → spotřebič
 5 DAgoto             then goto Konec;
 6 DAfast        určení rychlých hran;
 7 DAdeg         určení výchozích stupňů vrcholů;
 8 DAclea        pročištění sítě;
 9 DAsat         nalezení nasyceného toku;
10 DAback        goto Fáze;
11 DAend    Konec:  výpočet toků hranami z údajů o rezervách.
```

Dinitzův algoritmus

```
 1 DSwhil    while existuje hrana h taková, že h.status = použitelná do begin
 2 DSpath        nalezení cesty;
 3 DSproc        zpracování cesty;
 4 DSclea        pročištění sítě;
 5               end.
```

Nalezení nasyceného toku

Určení vrstev je jednoduchá aplikace prohledávání do šířky, vycházejíce ze zdroje; pro úplnost jej zde zopakujeme:

```
Proměnné:  Q, FIFO fronta vrcholů;
 1 LAinit     L(z) = 0; L(v) ← ∞ pro každý jiný vrchol v;
 2 LAiniQ     zařaď z do fronty Q;
 3 LAfor      for všechny hrany h do
 4 LAsat          if R(h) > 0 then h.status ← použitelná else h.status ← nasycená;
 5 LAwhil     while Q je neprázdná do begin
 6 LAoutQ         vyjmi první vrchol v z fronty Q;
 7 LAwfor         for všechny hrany h = (v, w) takové, že h.status = použitelná do
 8 LAinfi             if L(w) = ∞ then begin
 9 LAsetL                 L(w) ← L(v) + 1;
10 LAputQ                 zařaď w do fronty Q;
11                    end;
12               end;
```

Určení vrstev

Množiny vrcholů se stejnou hodnotou $L(v)$ budeme nazývat *vrstvy*.

Jestliže po určení vrstev je $L(s) = \infty$, pak ze zdroje do spotřebiče neexistuje nenasycená cesta a výpočet končí skokem na návěští "Konec", v opačném případě se pokračuje snadnou klasifikací nenasycených hran na rychlé a pomalé.

Nenasycená hrana (v, w) je rychlá právě když platí $L(v) < L(w)$. Hrana je tedy rychlá, jestliže postupuje o jednu vrstvu doprava.

Při vstupu do etapy klasifikace jsou všechny nenasycené hrany označeny jako použitelné, a v etapě klasifikace necháme všechny rychlé nenasycené hrany označeny jako použitelné (jsou to modré hrany z appletu), ale změníme statut hran pomalých. V etapě čistění mohou být některé použitelné hrany překlasifikovány na slepé, viz dále.

Klasifikace je využívána tak, že cesta pro zlepšení toku se zásadně vybírá z použitelných hran. Uvidíme, že se tím dosáhne dostatečné rychlosti výpočtu.

```
 1 DFfor     for všechny hrany h = (v, w) takové, že h.status = použitelná do
 2 DFif          if L(v) ≥ L(w)
 3 DFthen            then h.status ← pomalá;
```

Určení rychlých hran

Čištění sítě vychází z určení stupňů vrcholů, které je také jednoduché. V pseudokódu v appletu bylo implicitně zahrnuto do samotného pročišťování, zde jej uvedeme zvlášť. Číslo $D_{in}(v)$ resp. $D_{out}(v)$ bude udávat počet použitelných hran, vstupujících do vrcholu v, resp. vystupujících z vrcholu v. V průběhu také určíme pro nás nejzajímavější informaci - množiny M_{in} a M_{out} vrcholů, do kterých nevstupují resp. ze kterých nevystupují použitelné hrany.

```
1   for všechny vrcholy v do begin D_in(v) ← 0; D_out(v) ← 0; end;
2   for všechny hrany h = (u,v) takové, že h.status = použitelná do begin
3       zvyš D_in(v) o 1; zvyš D_out(u) o 1;
4       end;
5   M_in = ∅; M_out = ∅;
6   for všechny vnitřní vrcholy v do begin
7       if D_in(v) = 0 and D_out(v) ≠ 0 then vlož v do M_in;
8       if D_out(v) = 0 and D_in(v) ≠ 0 then vlož v do M_out;
9           then vlož v do M_out;
10      end;
```

Určení výchozích stupňů vrcholů

Pročištění sítě překlasifikuje některé hrany dosud klasifikované jako použitelné za slepé. Jak uvidíme ve škole, slepé hrany jsou ty, které není třeba uvažovat při hledání zlepšující cesty, protože přes ně nevede hledaná cesta minimální délky za zdroje do spotřebiče, složená jen z rychlých nenasycených hran. Pročištění sítě spočívá ve výstupním a vstupním pročistění sítě:

```
1 DCwhil   while M_out je neprázdná
2 DCoutM       vyjmi z M_out libovolný vrchol v;
3 DCfor        for každou hranu h = (u,v) vstupující do v
4 DCuse                takovou, že h.status = použitelná do begin
5 DCblnd           h.status ← slepá;
6 DCoutd           D_out(u) ← D_out(u) − 1;
7 DCind            D_in(v) ← D_in(v) − 1;
8 DCputM           if u není zdroj and D_out(u) = 0 and D_in(u) ≠ 0
9                      then vlož u do M_out;
10 DCend           end;
```

Výstupní pročištění sítě

```
1   while M_in je neprázdná
2       vyjmi z M_in libovolný vrchol u;
3       for každou hranu h = (u,v) vystupující z u
4               takovou, že h.status = použitelná do begin
5           h.status ← slepá;
6           D_out(u) ← D_out(u) − 1;
7           D_in(v) ← D_in(v) − 1;
8           if v není spotřebič and D_in(u) = 0 and D_out(u) ≠ 0
9               then vlož v do M_in;
10          end;
```

Vstupní pročištění sítě

```
Výstupní pročištění síté;
Vstupní pročištění síté;
```

Pročištění sítě

Jak bylo naznačeno již v procházce, proceduru označenou jako "Vstupní pročištění" bychom ve skutečnosti nemuseli používat, protože při hledání cesty se do hran, které jsou v ní upravovány, nedostaneme a proto ve skutečnosti konáme zbytečnou práci. Zkomplikovali bychom si tím ale

výklad algoritmu a dokazování jeho vlastností. Až jej ale budete programovat, můžete si ho v tomto smyslu upravit.

Nalezení nejkratší nenasycené cesty je velmi snadné, postupujeme stále vpřed použitelnými hranami a záhy se dostaneme do spotřebiče.

```
Proměnné:  vrchol v;
 1          v ← zdroj;
 2          while v ≠ spotřebič do begin
 3 DFedg        zvol hranu h = (v, w) vycházející z v
 4                  takovou, že h.status = použitelná;
 5              přidej hranu (v, w) do vytvářené cesty;
 6              v = w;
 7              end;
```

Nalezení cesty

Zpracování cesty využívá operace přesunu přebytku, jak byla definována v kapitole o FordFulkersonově algoritmu a to ve formulaci, která pracuje s hodnotami R a nikoli vlastními toky hran. Kromě vlastního přesunu přebytku ještě kontroluje, zda zpracovávané hrany zůstanou nenasycené a u těch, které přestanou být nenasycené (a tím i použitelné) se opraví informace o stupni počátku, který se popřípadě zařadí do seznamu slepých uliček.

```
Proměnné:  číslo Δ;
           stupně D_in(v) a D_out(v) vrcholů z Dinitzova algoritmu;
           M_in a M_out, seznamy slepých vrcholů z Dinitzova algoritmu;
 1          Δ ← ∞;
 2          for všechny hrany h cesty do if Δ > R(h) then Δ = R(h);
 3          for všechny hrany h cesty do begin
 4 DPdelt       přesuň přebytek Δ podél hrany h;
 5              if R(h) = 0 then begin;
 6 DPstat           h.status ← nasycená;
 7                  u ← počátek hrany h;
 8                  D_out(u) ← D_out(u) − 1;
 9                  if u není zdroj and D_out(u) = 0 and D_in(u) ≠ 0
10                      then zařaď' u do M_out;
11                  v ← konec hrany h;
12                  D_in(v) ← D_in(v) − 1;
13                  if v spotřebič and D_in(v) = 0 and D_out(v) ≠ 0
14                      then zařaď' v do M_in;
15                  end;
16 DPslow      if h^op.status = nasycená then h^op.status ← pomalá;
17             end;
```

Zpracování cesty

V uvedeném tvaru zpracování cesty udržuje aktuální stupně vrcholů i seznamy M_{in} a M_{out} neprůchozích vrcholů, proto pročištění sítě následující po zpracování cesty nemusí stupně a seznamy znovu určovat.

Abychom dodrželi požadavek nezáporného přebytku ve vnitřních vrcholech sítě, je třeba v druhém **for** cyklu probírat hrany v pořadí, ve kterém se nacházejí na cestě. Tento požadavek ale ve skutečnosti není pro správnost výpočtu nutný, hrany cesty můžeme probírat v libovolném pořadí.

8 Škola

Je zřejmé, že Dinitzův algoritmus je implementací Ford-Fulkersonova algoritmu; veškeré operace navíc jsou vedeny jediným cílem: vybrat cestu tak, aby to bylo rychlé a vedlo i k ukončení výpočtu

zpracováním malého počtu cest. Proto se nebudeme zabývat správností algoritmu, která vyplývá z úvah provedených v předchozí sekci, ale pouze dokážeme, že se vždy zastaví a to poměrně brzo. V této části budeme jako N označovat počet vrcholů sítě a jako M počet jejích hran.

Pro zjednodušení zápisu některých z následujících tvrzení zavedeme následující názvosloví, používající hodnot proměnných L, D_{in} a D_{out}:
hrana $h = (u, v)$ se nazývá dopředná, jestliže $L(w) = L(v) + 1$;
hrana $h = (u, v)$ se nazývá průchozí, jestliže buď u je zdroj nebo $D_{in}(u) \neq 0$ a současně v je spotřebič nebo $D_{out}(v) \neq 0$.

Výpočet Dinitzova algoritmu rozdělíme na fáze. Fáze je úsek výpočtu mezi dvěma po sobě jdoucími vstupy na návěští "Fáze" (řádka 2), neboli jedno provedení těla cyklu algoritmu. Mimo to výpočet zahrnuje jestě inicializaci na začátku a na konci určování vrstev, během něhož se zjistí, že neexistuje nenasycená cesta ze zdroje do spotřebiče s následným ukončením výpočtu vypočtením toku.

Každá fáze začíná určováním vrstev, které se v průběhu fáze provede jen jednou. Během určování vrstev se pro každý vrchol v určuje mimo jiné číslo $L(v)$ a tato čísla se pak až do konce fáze nemění. Důležitým údajem je číslo $L(s)$ (kde s znamená spotřebič), které budeme nazývat index fáze.

Nejdůležitější tvrzení o Dinitzově algoritmu jsou následující:

- index fáze roste, tedy index libovolné fáze, která není úvodní fází, je ostře větší než index fáze předchozí;
- v průběhu jednoho hledání nasyceného toku (řádka 9 algoritmu odpovídající proceduře "Nalezení nasyceného toku") použitelné hrany pouze ubývají a nemohou se objevit nové použitelné hrany; kromě toho zpracováním jedné cesty alespoň jedna hrana přestane být použitelnou a stane se nasycenou;
- v rámci jedné fáze je veškerá doba věnovaná čištění sítě úměrná počtu hran, které se při čištění stanou nepoužitelnými, tedy je $O(M)$.

Z těchto tvrzení snadno vyplyne odhad pro délku výpočtu: Je zřejmé, že index fáze je vždy roven délce nějaké prosté cesty ze zdroje do spotřebiče a proto je to celé kladné číslo nejvýše rovné $N-1$. Z prvního tvrzení tedy plyne, že během výpočtu nastane méně než N fází. Jedno zpracování cesty v rámci hledání nasyceného toku na základě druhého tvrzení sníží počet přípustných hran alespoň o 1 a tedy při jednom hledání nasyceného toku se zpracuje nejvíce M cest a za celý výpočet nejvíce NM cest. Jelikož zpracování jedné cesty u Dinitzova algoritmu trvá čas úměrný její délce, která je menší než N, zpracování cest zabere celkově čas $NMO(N) = O(N^2M)$. Podle třetího tvrzení strávíme čištěním sítě v jedné fázi čas $O(M)$ a tedy celkově ve všech méně než N fázích čas $O(NM)$. Jelikož ostatní činnosti jsou časově méně významné, dostaneme z toho odhad $O(N^2M)$ pro délku výpočtu Dinitzova algoritmu. Dinitzův algoritmus je tedy implementací Ford-Fulkersonova algoritmu se zaručenou krátkou dobou výpočtu. Jsou sice známy další vylepšení původního algoritmu, ale jsou logicky a programátorsky podstatně složitější a proto je neuvádíme.

Nyní výše uvedená tvrzení dokážeme a zopakujeme nastíněný důkaz výpočetní složitosti podrobně.

Je důležité si uvědomit, že toky hranami, resp. proměnné R měníme pouze použitím operace přesunutí přebytku v řádku 4 procedury "Zpracování cesty" a tedy na základě Tvrzení 5 z analýzy Ford-Fulkersonova algoritmu hodnota $R(h)$ stále udává rezervu hrany h.

Následující dvě tvrzení byla v zásadě dokázána v kapitole o prohledávání do šířky, proto je zde nedokazujeme.

Tvrzení 12 *[Labeled: DinitzShortest] V okamžiku ukončení procedury "Určení vrstev" pro každý vrchol v platí, že buď $L(v) = \infty$ a vrchol v je nedosažitelný ze zdroje nenasycenými hranami a nebo je $L(v)$ délka nejkratší cesty ze zdroje do v obsahující jen nenasycené hrany.*

Tvrzení 13 *[Labeled: DinitzL] V okamžiku ukončení procedury "Určení vrstev" pro každou hranu (v, w) platí, že buď $L(v) = \infty$ nebo $L(w) \leq L(v) + 1$.*

Druhé tvrzení platí proto, že pokud je v dosažitelný ze zdroje nenasycenými hranami (právě tehdy bude $L(v) < \infty$), pak se jednou dostane do fronty Q, a když z ní bude vyjímán, pak se buď nastaví hodnota $L(w)$ na $L(v)+1$ a nebo už hodnota $L(w)$ nastavena byla při probírání jiné hrany (u,w), kde u vstoupil do fronty dříve než v a byl z ní proto také dříve vyjmut; jelikož při prohledávání do šířky hodnoty proměnných L vrcholů seřazených podle toho, jak byly zařazovány do fronty, tvoří neklesající posloupnost, je pak $L(u) \leq L(v)$ a tedy $L(w) = L(u)+1 \leq L(v)+1$.

Poznamenejme také, že v dané fázi se hodnoty proměnných L po ukončení procedury "Určení vrstev" nikde nemění.

Důležitým, i když prostým, důsledkem Tvrzení 13 je

Tvrzení 14 *[Labeled: DinitzInterLayer] Jsou-li u a v dva vrcholy, pak žádná cesta z u do v obsahující jen nenasycené hrany není kratší než $L(v) - L(u)$.*

Následující triviální tvrzení popisuje status hran po ukončení procedury "Určení vrstev":

Tvrzení 15 *[Labeled: DinitzSat] Po ukončení procedury "Určení vrstev" (řádek 3 Dinitzova algoritmu) pro každou hranu h platí, že je buď nasycená je nasycená nebo použitelná, a je nasycená právě když $R(h) > 0$.*

Důkaz: Pro každou hranu se provede řádek 4 procedury "Určení vrstev", kde se status hrany nastaví, jak udává tvrzení. Dále se už v této proceduře status hrany nikde nemění. ♣

"Určení vrstev" pouze rozdělí hrany na nasycené a ostatní, které jsou prozatímně označené jako použitelné. Teprve procedura "Určení rychlých hran" vezme v úvahu rozdělení vrcholů do vrstev podle hodnot L a překlasifikuje hrany zatím označené jako použitelné (a tedy považované za rychlé) na hrany pomalé. Podívejme se nyní na některé důsledky pro zbývající (a hlavní) část fáze.

Pouhým prohlédnutím prováděných změn stavů hran zjistíme, že

Tvrzení 16 *[Labeled: DinitzStatusChange] Nechť h je hrana. Od ukončení procedury "Určení rychlých hran" (řádek 6 Dinitzova algoritmu) do konce fáze mohou nastat pouze tyto změny stavu hrany h:*
z použitelné na slepou
z použitelné na nasycenou
z nasycené na pomalou.

Z tohoto tvrzení plyne důležitý důsledek, že od ukončení procedury "Určení rychlých hran" do konce fáze použitelných hran může jen ubývat. Plyne z toho také jednoduché

Tvrzení 17 *[Labeled: DinitzOpp] Od ukončení procedury "Určení rychlých hran" do konce fáze platí, že hrana opačná k použitelné hraně není dopředná.*

Důkaz: Hrana $h = (u,v)$, která je použitelná kdykoli při provádění řádek 7 až 10 Dinitzova algoritmu musela podle předchozího tvrzení být použitelná bezprostředně po ukončení "Určení rychlých hran" a tehdy byla určitě dopředná, protože jinak by byla překlasifikována na pomalou. Platilo tedy $L(v) = L(u)+1$, a jelikož se hodnoty $L(u)$ a $L(v)$ do konce fáze nemění, platí po celou uvažovanou dobu, že $L(u) < L(u)+2 = (L(u)+1)+1 = L(v)+1$. Tato nerovnost ale znamená, že hrana (v,u) opačná k h není dopředná. ♣

Tvrzení 18 *[Labeled: DinitzFast] Od ukončení procedury "Určení rychlých hran" do konce fáze pro každou hranu $h = (v,w)$ platí, že*
(a) buď h je nasycená a $R(h) = 0$;
(b) nebo h je pomalá a $R(h) > 0$ a h není dopředná;
(c) nebo h je použitelná nebo slepá a $R(h) > 0$ a h je dopředná.

Důkaz: "Určení rychlých hran" nemění rozdělení hran na nasycené a ostatní podle hodnot R. Hrany, které jsou nenasycené rozdělí na (stále) použitelné a pomalé na základě toho, zda jsou dopředné, což je podmínka testovaná v řádku 2 "Určení rychlých hran". Tvrzení tedy platí bezprostředně po ukončení procedury a nyní dokážeme, že až do konce fáze se jeho platnost neporuší.

Dopřednost hrany je určena jen hodnotami L pro její konce, které se od ukončení "Určení vrstev" až do konce fáze nemění. V procedurách "Pročištění sítě" a "Nalezení cesty" se nemění ani hodnoty R hran, ani jejich rozdělení na nasycené, pomalé a rychlé, jen se při pročištění může status hrany měnit z použitelné na slepou.

Platnost tvrzení by tedy pro hranu h mohla být ohrožena jen v proceduře "Zpracování cesty" jako důsledek změny $R(h)$ nebo $R(h^{op})$ při převedení přebytku v řádku 4. Pro hranu h hodnota $R(h)$ klesne. Může tedy klesnout na 0, ale v tom případě je skutečně hrana vzápětí překlasifikována na nasycenou v řádku 6, aby se dokazované tvrzení neporušilo, jen namísto (c) bude platit (a).

V řádku 4 byla hrana h určitě použitelná, takže podle Tvrzení 17 h^{op} není dopředná. Pro hranu h^{op} hodnota $R(h^{op})$ v řádku 4 stoupne. Pokud již byla nenulová, tedy h^{op} nebyla nasycená, nenulovost $R(h^{op})$ se nemění a status h^{op} se nemění - podmínka v řádku 16 pro ni není splněna.

Pokud $R(h^{op})$ bylo rovno nule a tedy hrana h^{op} byla nasycená, pak vzrůst $R(h^{op})$ znamená jeho změnu na kladnou hodnotu. Vzápětí, v řádku 16, je h^{op} překlasifikována na pomalou a navíc již víme, že není dopředná. Bude proto pro ní platit možnost (b) namísto původní možnosti (a). ♣

Nyní se budeme zabývat rychlými hranami a konkrétně jejich rozdělení na použitelné a slepé, o které se starají procedury pročištění. Datové struktury, které jsou pro to využívány jsou proměnné $D_{in}(u)$, $D_{out}(u)$, a M_{in} a M_{out}.

Tvrzení 19 *[Labeled: DinitzDeg] V řádcích 8 a 9 Dinitzova algoritmu pro každý vrchol u platí, že*
$D_{in}(u)$ je počet použitelných hran vstupujících do vrcholu u,
$D_{out}(u)$ je počet použitelných hran vystupujících z vrcholu u,
M_{in} je množina vnitřních vrcholů u, pro které je $D_{in}(u) = 0$ a současně $D_{out}(u) \neq 0$,
M_{out} je množina vnitřních vrcholů u, pro které je $D_{out}(u) = 0$ a současně $D_{in}(u) \neq 0$.

Důkaz: Procedura "Určení výchozích stupňů" vrcholů je zjevně provedena přesně tak, aby po jejím dokončení tvrzení platilo.

Změny proměnných vyskytujících se v tvrzení probíhají při pročišťování sítě. Proberme výstupní čištění, pro vstupní je důkaz obdobný. Je zřejmé, že kdykoli se některá hrana překlasifikuje z použitelné na slepou, sníží se o 1 příslušné stupně jejích konců tak, aby se platnost prvních dvou řádků tvrzení neporušila. Pro vrchol v, který byl vyjmut z M_{out} platilo $D_{out}(v) = 0$ a současně $D_{in}(v) \neq 0$, ale vzápětí se všechny přijatelné hrany vstupující do v překlasifikují na slepé a s tím také $D_{in}(v)$ klesne na 0, což spolu s tím, že $D_{out}(u)$ je stále 0, odpovídá tomu, že v byl vyjmut z M_{out}. Nakonec pokles $D_{out}(u)$ v řádku 6 procedury může způsobit, že tato hodnota poklesne na nulu a pak se popřípadě vrchol zařadí do M_{out} tak, aby platnost dokazovaného tvrzení nebyla narušena.

Hodnoty proměnných, vyskytujících se v tvrzení, se nemění při hledání cesty, ale jejich změny probíhají při zpracování cesty. I tam však jsou stupně koncových vrcholů hrany nebo hran, které ze z použitelných změní na nasycené, mění tak, aby vyjadřovaly počet incidentních použitelných hran, a patřičným způsobem se aktualizují i množiny M_{out} a M_{in}. ♣

Nyní již je možno dokázat hlavní charakterizační tvrzení pro hrany, které říká, že statut hrany algoritmus nastavuje tak, že jednoduše souvisí s její nasyceností, dopředností a průchodností.

Tvrzení 20 *[Labeled: DinitzClassif] Při vstupu do procedury "Nalezení cesty" v řádku 2 procedury "Nalezení nasyceného toku" (tedy vždy po ukončení procedury "Pročištění sítě" v řádku 8 základního algoritmu nebo řádku 4 procedury "Nalezení nasyceného toku") pro každou hranu $h = (u, v)$ platí, že*

buď h je nasycená a $R(h) = 0$;
nebo h je pomalá a $R(h) > 0$ a h není dopředná;
nebo h je použitelná a $R(h) > 0$ a h je dopředná a průchozí;
nebo h je slepá a $R(h) > 0$ a h je dopředná, ale není průchozí.

Důkaz: Na základě předchozích tvrzení stačí dokázat, že dopředné hrany s kladnou rezervou jsou rozděleny na použitelné a slepé na základě toho, zda jsou průchozí.

Tvrzení 16 říká, že nemůže tedy vzniknout nová rychlá (t.j. použitelná nebo slepá) hrana a pokud by se některá rychlá hrana stala nasycenou nebo pomalou, již nás nemusí zajímat.

Při pročišťování stupně D_{in} a D_{out} mohou jen klesat, takže pokud pro nějaký vrchol u je $D_{in} = 0$ nebo $D_{out} = 0$, pak se to během pročišťování nemůže změnit. Hrana, která není průchozí, se tedy při pročišťování průchozí nemůže stát.

Pokud by nějaká hrana $h = (u, v)$ byla na začátku "Pročištění sítě" použitelná, ale nikoli průchozí, pak by platilo, že buď u není zdroj a přitom $D_{in}(u) = 0$ nebo v není spotřebič a přitom $D_{out}(v) = 0$. Díky hraně h však $D_{out}(u) > 0$ a $D_{in}(v) > 0$, takže na základě Tvrzení 19 by buď u patřil do M_{in} nebo v by patřil do M_{out}. V průběhu pročišťování sítě by pak buď u byl vyjmut z M_{in} nebo v by byl vyjmut z M_{out}; v obou případech by h byla překlasifikována na slepou, zůstávajíc neprůchozí.

Po určení vrstev a stupňů nejsou slepé hrany; slepou se hrana $h = (u, v)$ může stát změnou z použitelné. V okamžiku, kdy se tak děje musí být buď $u \in M_{in}$ nebo $v \in M_{out}$, takže v ten okamžik není průchozí a neprůchozí zůstane až do konce čištění. Slepá průchozí hrana tedy nemůže vzniknout. ♣

Pro zjednodušení formulace následujících tvrzení budeme říkat, že cesta v síti je použitelná, pokud se skládá výhradně z použitelných hran.

Nyní ukážeme, že po pročištění sítě lze použitelnou cestu pro zlepšení toku hledat jednoduchou a rychlou procedurou "Nalezení cesty", která nepřipouští žádné návraty zpět ze slepých uliček. Klíčové je následující

Tvrzení 21 *[Labeled: DinitzPath] Při vstupu do procedury "Nalezení cesty" v řádku 2 procedury "Nalezení nasyceného toku" (tedy vždy po ukončení procedury "Pročištění sítě" v řádku 8 základního algoritmu nebo řádku 4 procedury "Nalezení nasyceného toku") pro každou hranu $h = (u, v)$ platí, že h je použitelná právě když přes ní prochází alespoň jedna použitelná cesta ze zdroje z do spotřebiče s.*

Důkaz: Nechť hrana h je po ukončení pročišťování použitelná. Nechť $h_1, ..., h_\ell$ je nejdelší cesta složená z použitelných hran, která obsahuje hranu h (tedy $h = h_i$ pro nějaké i takové, že $1 \leq i \leq \ell$). Označme začátek h_0 jako u_0. Jelikož h_0 je použitelná a tedy podle předchozího tvrzení průchozí, pak kdyby u_0 nebyl zdroj, muselo by být $D_{in}(u_0) > 0$, tedy podle Tvrzení 19 by do u_0 vstupovala použitelná hrana, což je spor s tím, že cesta $h_0, ..., h_\ell$ je nejdelší. Podobně by se dostal spor s předpokladem, že poslední hrana cesty nekončí ve spotřebiči. Cesta je tedy začíná ve zdroji a končí ve spotřebiči. ♣

Tvrzení 22 *[Labeled: DinitzPathL] Použitelná cesta ze zdroje do spotřebiče má vždy délku $L(s)$.*

Důkaz: Nechť $z = v_0, v_1, ..., v_\ell = s$ jsou vrcholy použitelné cesty ze zdroje do spotřebiče, tedy pro $i = 1, ..., \ell$ je (v_{i_1}, v_i) použitelná hrana. To ale implikuje, že pro všechna taková i je $L(v_{i_1}) + 1 = L(v_i)$ a jelikož $L(v_0) = L(z) = 0$, plyne z toho $L(v_i) = i$. Z toho ale pro délku ℓ cesty vyplývá, že $\ell = L(v_\ell) = L(s)$. ♣

Tvrzení 23 *[Labeled: DinitzFindPath] Procedura "Nalezení cesty" vždy nalezne použitelnou cestu ze zdroje z do spotřebiče s.*

Důkaz: V řádku 3 procedury má algoritmus zvolit použitelnou hranu vycházející z vrcholu v a především musíme dokázet, že takový hrana vždy existuje.

Jestliže existuje alespoň jedna použitelná hrana (což je podmínka pro vstup do procedury "Nalezení cesty"), pak podle předchozího tvrzení přes ni jde cesta ze zdroje do spotřebiče, složená z použitelných hran, což implikuje, že ze zdroje z vychází alespoň jedna použitelná hrana a tedy pro $v = z$ potřebná použitelná hrana existuje.

V okamžiku, kdy v už není zdroj, je v okamžitý konec vytvářené cesty a tedy do něj vstupuje použitelná hrana h - ta, která je poslední v již vytvořeném segmentu cesty. Podle Tvrzení 21 tedy hranou v prochází použitelná cesta $\mathcal{C}$ směřující do spotřebiče. Pokud tedy v není spotřebič (což je podmínka pro to, abychom se znovu dostali do řádku 3), musí z v vycházet alespoň jedna použitelná hrana (přinejmenším hrana cesty $\mathcal{C}$ následující za hranou h) a tedy volba v dalším provedení řádku 3 je možná. ♣

Poznamenjme, že cesta vytvářená procedurou "Nalezení cesty" se musí "trefit" do spotřebiče. Použitelná cesta, která by jej minula, by musela postupovat stále dál k vyšším vrstvám vrcholů, takže by nakonec skončila ve slepé uličce, takže by byla před hledáním cesty odstraněna alespoň částečně při pročišťování, které hledání cesty bezprostředně předchází. Použitelné cesty, které zbudou po pročištění tedy všechny směřují do spotřebiče.

Řekneme, že cesta je nenasycená, jestliže neobsahuje žádnou nasycenou hranu.

Následující tvrzení je svým způsobem obrácením Tvrzení 22.

Tvrzení 24 *[Labeled: DinitzShortPath] Nenasycená cesta ze zdroje do spotřebiče, jejíž délka je rovna indexu fáze je použitelná.*

Důkaz: Nechť $z = u_0, \ldots, u_\ell = s$ jsou vrcholy takové, že pro $i = 1, \ldots, \ell$ je (u_{i-1}, u_i) nenasycená hrana a předpokládejme, že $\ell = L(s)$. Je $L(u_0) = L(z) = 0$ a podle Tvrzení 13 je také $L(u_{i-1})+1 \leq L(u_i)$ pro každé i, což implikuje $L(i) \leq i$. Kdyby pro nějaké i platila nerovnost, pak by bylo také $L(u_\ell) < \ell = L(s)$, spor. Tedy všechny hrany cesty musí být dopředné.

Existence celé cesty ale také implikuje, že stupně D_{in} a D_{out} pro všechny vrcholy cesty jsou (s případnými jednostrannými výjimkami pro zdroj a spotřebič) kladné a tedy všechny hrany cesty jsou průchozí na základě Tvrzení 19 a proto musí být použitelné podle Tvrzení 20. ♣

Teď se již dostáváme k závěrečnému odhadu časové složitosti. Již víme, že v rámci jedné fáze použitelné hrany nepřibývají, ale

Tvrzení 25 *[Labeled: DinitzLess] Procedura "Nalezení cesty" zmenší počet použitelných hran.*

Důkaz: Alespoň pro jednu hranu h ze zpracovávané cesty musí platit $R(h) = \Delta$. Pro tuto hranu h přesun přebytku Δ znamená zmenšení $R(h)$ na 0 s následným překlasifikováním z použitelné na nasycenou. V rámci procedury se použitelnou nemůže stát hrana, která použitelná nebyla. ♣

Všechna výše dokázaná tvrzení o hodnotách proměnných se týkala provádění jedné fáze a byla formulována s podstatným využitím proměnných L, které se v následující fázi mohou zcela změnit. Následující tvrzení ale je nezávislé na hodnotách specifických pro danou fázi a váže k sobě dvě po sobě následující fáze. Využíváme při něm Tvrzení 24, které charakterizuje použitelné cesty ze zdroje do spotřebiče pouze s použitím pojmu nasycenosti hrany, který je nezávislý na proměnných vázaných na jednu fázi (s výjimkou indexu fáze).

Tvrzení 26 *[Labeled: DinitzIndex] Označíme-li délku nejkratší nenasycené cesty ze zdroje do spotřebiče při k-tém vstupu do řádku 2 Dinitzova algoritmu jako $\mathcal{L}_k$ (a $\mathcal{L}_k = N$, kde N je počet vrcholů sítě, pokud taková cesta neexistuje), pak vstoupí-li výpočet do řádku 2 algoritmu q-krát, pak $\mathcal{L}_1 < \mathcal{L}_2 < \cdots < \mathcal{L}_q$.*

Důkaz: Označme v průběhu jedné fáze jako M množinu všech nenasycených cest ze zdroje do spotřebiče, které mají délku rovnou nebo menší než je index fáze. Pro důkaz tvrzení stačí dokázat, že na začátku fáze je M neprázdná, ale na jejím konci je prázdná.

Z Tvrzení 13 plyne, že "Určení vrstev" nastaví $L(s)$ (které udává index fáze) na hodnotu rovnou délce nejkratší nenasycené cesty. Z toho plyne, že na začátku fáze (která není předčasně ukončena skokem na Konec) existuje nenasycená cesta ze zdroje do spotřebiče délky rovné indexu fáze, takže M je neprázdná.

Kdyby na konci fáze byla M neprázdná, existovala by nenasycená cesta ze zdroje do spořebiče délky rovné indexu fáze. Ta by ale byla podle Tvrzení 24 použitelná a proto by v síti existovala použitelná hrana. Fáze ale může skončit jen pokud v síti žádná použitelná hrana není. ♣

Právě dokázané tvrzení je nejdůležitějším výsledkem o časové složitosti Dinitzova algoritmu a ihned z něho plyne

Tvrzení 27 *[Labeled: DinitzIndexN] Počet fází výpočtu podle Dinitzova algoritmu není větší než počet vrcholů sítě.*

Důkaz: Nejkratší nenasycená cesta ze zdroje do spotřebiče musí být prostá (žádný vrchol se na ní nevyskytuje vícekrát) a proto obsahuje nejvíce $N-1$ hran, kde N je počet vrcholů sítě. Index sítě je tedy podle Tvrzení 24 nejvýše $N-1$. ♣

Výše uvedená tvrzení také umožňují odhadnout dobu, potřebnou k provedení jedné fáze.

Tvrzení 28 *[Labeled: DinitzPathRpt] V jedné fázi Dinitzova algoritmu se provede "Nalezení cesty" i "Zpracování cesty" nejvýše M-krát, kde M je počet hran sítě.*

Důkaz: Podle Tvrzení 25 a Tvrzení 16 operace "Pročištění sítě", "Nalezení cesty" i "Zpracování cesty" nezvětšují počet použitelných hran a posledně jmenovaná procedura ho při každém provedení zmanší alespoň o 1. Počet provedení "Zpracování cesty" v rámci jedné fáze tedy nemůže být větší než byl výchozí počet použitelných hran v této fázi po provedení "Určení vrstev", což je ovšem nejvýše M. ♣

Tvrzení 29 *[Labeled: DinitzTimePath] Jedno provedení procedury "Nalezení cesty" spolu s následným "Zpracováním cesty" provedou $O(N)$ kroků, kde N je počet vrcholů sítě.*

Důkaz: Jak "Nalezení cesty", tak i "Zpracování cesty" trvají čas, který je úměrný délce nalezené, respektive zpracovávané cesty a tato délka nemůže být větší než N (?). ♣

Tvrzení 30 *[Labeled: DinitzOnceInM] Jestliže byl vrchol u vyjmut z M_{out} (resp. z M_{in}), pak se v průběhu dané fáze do M_{out} (resp. do M_{in}) znovu nedostane.*

Důkaz: Z Tvrzení 19 plyne, že u je v M_{out} pouze když se nejedná o zdroj a $D_{out}(u) = 0$. Jestliže je z M_{out} vyjmut, stane se to v řádku 2 výstupního pročištění sítě a v následujícím provedení **for** cyklu v řádcích 3 a 10 této procedury se všechny hrany vstupující do u překlasifikují na slepé, takže pak bude $D_{in}(u) = 0$ a protože $D_{in}(u)$ neroste, tento stav se do konce fáze nezmění. Proto již nikdy v dané fázi nebude zařazen do M_{out}, protože nebude splněna podmínka řádku 8 "Výstupního pročištění sítě", což je jediné místo, kde by se ještě do M_{out} mohl dostat. Pro M_{in} je důkaz obdobný. ♣

Tvrzení 31 *[Labeled: DinitzTimeClean] V jedné fázi Dinitzova algoritmu se dohromady ve všech provedeních procedury Pročištění sítě provede $O(M)$ kroků, kde M je počet hran sítě.*

Důkaz: Z předchozího Tvrzení plyne, že řádky 1 a 2 "Výstupního pročištění sítě" se nemohou v rámci jedné fáze provést vícekrát, než kolik je vrcholů sítě - každé provedení je vázáno ke konkrétnímu vrcholu, který je vyjímán z M_{out}. Jelikož jedno provedení kteréhokoli z řádků 2, 5-8 trvá konstantní čas, nezávislý na velikosti sítě, pak jedno provedení **for** cyklu v řádcích 3-10 trvá čas, který je v nejhorším případě úměrné stupni tohoto vrcholu. Proto se tělo cyklu neprovede vícekrát než kolik byl výchozí počet použitelných hran v síti v uvažované fázi, tedy nejvýše M-krát. Podobná úvaha se provede pro Vstupní pročištění sítě. ♣

Tvrzení 32 *[Labeled: DinitzTimePhase] Má-li síť S celkově N vrcholů a M hran, pak jedna fáze Dinitzova algoritmu skončí po $O(NM)$ krocích.*

Důkaz: "Určení vrstev", "Určení rychlých hran" a "Určení výchozích stupňů" zjevně trvá čas nejvýše úměrný součtu počtu vrcholů a hran, tedy výrazně méně, než je potřeba pro odhad $O(NM)$.

Celkový počet kroků při pročišťování je v jedné fázi podle Tvrzení 31 $O(M)$. Nakonec určování a zpracovávání cest se provede nejvýše M-krát a každé vyžaduje čas $O(N)$, viz Tvrzení 28 a Tvrzení 29. ♣

Věta 33 *Má-li síť S celkově N vrcholů a M hran, pak Dinitzův algoritmus nalezne největší tok v síti S po $O(N^2M)$ krocích.*

Důkaz: Věta je snadnou kombinací časového odhadu pro jednu fázi a odhadu pro počet vykonaných fází, protože řádky 1 a 11 algoritmu (inicializace a ukončení, které nejsou zahrnuty do fází) vyžadují čas úměrný počtu vrcholů a hran sítě. ♣

9 Goldbergův algoritmus

Ford-Fulkersonův algoritmus a řada dalších algoritmů, které z něho byly odvozeny, jako je Dinitzův algoritmus, jsou představitelé jednoho typu algoritmů pro toky v sítích, které využívají zlepšující cesty. V této části popíšeme jiný algoritmus, který je základním představitelem algoritmů pracujících s nehotovým tokem.

Zatímco metoda zlepšující cesty by se dala snadno vysvětlit bez použití pojmu přebytku toku ve vrcholu a ve Ford-Fulkersonově algoritmu i jeho implementacích, jako je Dinitzův algoritmus, se přebytky objeví jen na velmi krátkou dobu a vždy jen v jediném vrcholu, metoda nehotového toku předpokládá, že po celou dobu výpočtu je v alespoň jednom a většinou v mnoha vnitřních vrcholech kladný přebytek. Budeme ale vyžadovat, aby žádný vrchol (s výjimkou zdroje) neměl nikdy záporný přebytek neboli deficit.

V Goldbergově algoritmu má každý vrchol *výšku*, což je celé nezáporné číslo. Výšku vrcholu v budeme označovat $H(v)$.

Inicializace algoritmu nastaví výšku zdroje na hodnotu rovnou počtu vrcholů sítě, výšky ostatních vrcholů na 0; přebytky všech vrcholů jsou rovny nule, toky hranami jsou nulové a rezervy všech hran jsou rovny jejich kapacitám.

Vytvoření počátečního nehotového toku převede každou hranou vycházející ze zdroje přebytek rovný kapacitě této hrany. Rezervy těchto hran tím poklesnou na nulu.

Algoritmus potom probírá vnitřní vrcholy sítě, které mají kladný přebytek; jestliže se pro takový vrchol v najde z něho vycházející hrana, která má kladnou rezervu a *směřuje dolů* (t.j. její konec má menší výšku než její počátek v), pak hranou převedeme tolik přebytku, kolik je minimum přebytku vrcholu v a rezervy hrany. Nemůžeme totiž převádět více přebytku než ve vrcholu je, protože chceme aby přebytky byly nezáporné, a z kapacitních důvodů nelze převádět více přebytku než je rezerva hrany. Minimum přebytku vrcholu a rezervy hrany je tedy největší množství přebytku, které lze převést.

V okamžiku, kdy již v grafu není žádný vnitřní vrchol s nenulovým přebytkem, výpočet končí.

Algoritmus je mimořádně jednoduchý a zdá se být i přirozený v tom, že přebytek odtéká "z kopce" a jelikož zdroje je výše než spotřebič, tak tok bude téci v požadovaném směru. Bystrý čtenář možná přijde i na to, že pokud se veškerý tok představující počáteční nehotový tok nepodaří protlačit do spotřebiče, vrcholy ve kterých se zadrhne nakonec vystoupají tak vysoko, že z nich přebytky odtečou zpět do zdroje a v žádném vrcholu nezbyde kladný přebytek. Nakonec je jasné, že výpočet skončí, když v žádném vnitřním vrcholu sítě nebude nenulový přebytek, takže na konci výpočtu algoritmus vytvoří tok.

Zřejmé naopak vůbec není, že by se algoritmus měl vždy zastavit. Kromě toho není patrno, proč by výsledný tok měl mít největší možnou velikost. Jak uvidíme, kdyby zdroj na začátku byl sice vyzvednut, ale do výšky menší než je počet vrcholů, mohlo by se stát, že by výsledný tok skutečně nebyl optimální, což svědčí o tom, že důvody proč algoritmus dá požadovaný výsledek nejsou zcela triviální.

Scéna: *Jednoduchá cesta*

Pro pochopení činnosti Goldbergova algoritmu je velmi užitečné ukázat si jeho výpočet na několika jednoduchých případech. Podobně jako dříve bude přebytek ve vrcholu znázorňován sloupečkem odpovídající výšky.

První případ je síť představovaná prostou orientovanou cestou, jejíž všechny hrany mají stejné kapacity, v tomto případě rovné 1. Vzhledem k linearitě sítě je snadné obrázek nakreslit v 2D i s vystižením výšek vrcholů, které odpovídají výšce na obrazovce.

Procházejte si výpočtem krok za krokem pomocí knoflíku [**Krok**] a sledujte průběh výpočtu.

V kroku vytvoření počátečního nehotového toku se (jedinou) hranou vycházející ze zdroje přenese do prvního vnitřního vrcholu sítě přebytek rovný kapacitě hrany, v našem případě velikosti 1, zdroj má deficit 1. Hrana se tím nasytí (t.j. její rezerva klesne na 0), opačná hrana získá rezervu 1.

V následujícím kroku má kladný přebytek pouze první vnitřní vrchol, ale hrana z něj jdoucí vlevo strmě stoupá vzhůru a hrana vpravo je vodorovná, proto přebytek nelze převést. Podle algoritmu je tedy vrchol zvednut.

Po zvednutí hrana vedoucí z vrcholu doprava klesá a má dostatečnou rezervu a proto se jí převede přebytek o jeden vrchol doprava. Druhý vrchol musí být nejprve zvednut, pak postoupí přebytek dále doprava a tímto způsobem postupuje, dokud nedoteče do spotřebiče. Vzhledem ke kapacitám hran se pokaždé podaří převést přebytek celý a proto má výsledný tok velikost rovnu kapacitě první hrany (té která vychází ze zdroje).

Scéna: *Cesta s překážkou*

Tato síť zjevně připouští pouze tok velikosti 1, protože pravou hranou více toku nepřejde, ale na počátku ze zdroje vytéká levou hranou tok velikosti 2, který vytvoří přebytek velikosti 2 v prostředním vrcholu. Střední vrchol musí být nejprve zvednut, pak z něj může odtéci část přebytku do spotřebiče.

Pravou hranou ale odteče jen polovina přebytku, druhá ve vrcholu zůstane a nemůže odtéci ani vpravo, kde je hrana již natrvalo nasycena, ani vlevo, kde hrana jde prudce do kopce.

Střední vrchol proto stoupá vzhůru. V okamžiku, kdy se dostane o jednu hladinu nad zdroj, pak všechen přebytek odteče vlevo zpět do zdroje, protože hrana směřující vlevo dolů má patřičnou kapacitu.

Tato scéna ukazuje základní mechanismus, jak odstranit tu část počátečního nehotového toku, kterou se nepodaří dopravit do spotřebiče.

Scéna: *Dlouhá cesta s překážkou*

Tato scéna je podobná předchozí, ale síť je tvořena dlouhou cestou, ve které kapacita poslední hrany je menší než jsou kapacity hran předchozích.

Výpočet probíhá podobně; počáteční vlna postupně dorazí až těsně před spotřebič, kde její část projde do cíle, ale další část se odrazí zpět. Způsob jak stoupají vzhůru vrcholy oblasti, ve které

se zbytkový tok přelévá z jedné strany na druhou, je nyní podstatně zdlouhavější a složitější než v předchozí scéně, ale zásadní odlišnost zde nevzniká.

Způsob zvedání oblasti bez dočasného odtoku by si zasloužil zlepšení, ale při analýze algoritmu později uvidíme, že obecně výpočetnímu času dominuje doba věnovaná nenasyceným převodům přebytku, takže úpravy způsobu zvedání vrcholů, které by mohly algoritmus značně zkomplikovat s nepříliš jistým efektem na celkovou výpočetní dobu, se obvykle neprovádějí.

Scéna: *Cesta s překážkami*

Scéna se podobá předchozí, ale míst zúžení cesty je více a nesousedí přímo se spotřebičem. V tomto případě vznikne více vrcholů s kladným přebytkem a proto applet implementuje několik různých strategií, jak vybrat jeden z nich, protože výše uvedené schéma výběr nespecifikuje.

V závislosti na ovladači bude z vnitřních vrcholů s kladným přebytkem vybrán nejstarší, nejmladší, nejvyšší, nejnižší nebo náhodně zvolený (a v případě výběru nejvyššího a nejnižšího se v případě neurčitosti vybere libovolný z kandidátů s extrémní výškou).

Je vidět, že i když výpočet obecně pro jednotlivé volby probíhá dosti odlišně, žádná z možností nepřináší rozhodující výhody (přinejmenším v tomto příkladě).

Scéna: *Síť s propustnými větvemi*

V této scéně síť není prostá cesta, ale ve vrcholu uprostřed se rozdělí do dvou větví. Je použito 3D zobrazení. Pomocí myši je možno síť natáčet a naklápět (stiskněte knoflík myši a tahněte jí vodorovně nebo svisle).

Tok vypuštěný ze zdroje napřed vstoupí do jedné větve, pak se jeho část odrazí zpět, v rozvětvení vrácený podíl toku vstoupí do druhé větve a je úspěšně dopraven do spotřebiče. Jelikož vrchol rozvětvení byl dosti nízko vzhledem ke zdroji, nemohlo dojít k navrácení toku, který neprošel první větví, zpět do zdroje, ale vše vstoupilo do druhé větve.

V této scéně byl vždy jediný vnitřní vrchol s nenulovým přebytkem a proto není rozdíl mezi variantami výběru vrcholu s nenulovým přebytkem.

Scéna: *Síť s polopropustnými větvemi*

Scéna se podobá předchozí, ale větve nemají ani dohromady dostatek kapacity pro převedení počátečního toku do spotřebiče. Tok, který se nepodařilo odvést do spotřebiče je proto nakonec vrácen do zdroje.

Scéna: *Síť se slabými větvemi*

Scéna se zase podobá předchozí, ale kapacity hran směrem ke spotřebiči klesají, proto se do vrcholů dostává více nehotového toku, než je možno odvést. Celkový průběh výpočtu je v zásadě podobný předchozí scéně, ale postupně obecně vzniká velké množství vrcholů s přebytkem. Různé varianty volby vrcholu s přebytkem proto jsou dosti odlišné. Zkuste si výpočet pro všechny varianty (volba na ovladači) a sledujte rozdíly mezi variantami.

Scéna: *Výběr dolního vrcholu*

Jak bylo řečeno, Goldbergův algoritmus nepředepisuje, který vrchol s přebytkem se vybírá. Nejlepší známý odhad časové složitosti v nejhorším případě má varianta, která vždy vybírá nejvyšší vrchol (nebo libovolný z nejvyšších). V této a následující scéně ukážeme, jakou výhodu má volba nejvyššího vrcholu před volbou nejnižšího vrcholu a dalšími variantami algoritmu.

Aby bylo možné ukázat rozdíly variant na co nejjednodušší síti, zvolíme poněkud násilný postup. Ze začátku se vrcholy s přebytkem volí nikoli jako nejvyšší nebo nejnižší, ale tak, aby se vytvořila konfigurace, na které bude rozdíl mezi variantami velmi výrazný. Od okamžiku vytvoření konfiguace se již bude volit buď vždy nejnižší vrchol (v této scéně) nebo vždy nejvyšší vrchol (v následující scéně).

Prvním klepnutím na knoflík [**Krok**] se neprovede jediný krok, ale série kroků, která vytvoří výše zmíněnou konfiguraci. Pokud se ale na ovladači zvolí [**Krátký krok**] nebo [**Střední krok**]

namísto [**Dlouhý krok**], pak je možno sledovat, jak se do konfigurace dostaneme (v případě krátkého kroku to bude trvat dosti dlouho). Od okamžiku vytvoření konfigurace pak stisknutí knoflíku [**Krok**] provede jediný přenos přebytku nebo zvednutí vrcholu, bez ohledu na nastavení délky kroku.

Stiskněte tedy knoflík [**Krok**] a tím vytvoříte výše zmíněnou konfiguraci, kdy každý vnitřní vrchol má kladný přebytek velikosti 1 a tyto vrcholy vytvářejí cestu končící ve spotřebiči s klesající výškou vrcholů. Varianta volící nejnižší vrchol převádí přebytek každého vrcholu do spotřebiče zvlášť cestou, která může být i velmi dlouhá. Myslím, že celý výpočet nedokončíte, protože trvá hodně dlouho, ale jistě pochopíte proč je tak pomalý podstatně dříve, než bude ukončen.

Scéna: *Výběr horního vrcholu*

Zde opakujeme výpočet předchozí scény s tím, že z uvedené předělové konfigurace postupujeme tak, že vybíráme vždy nejvšší vrchol s přebytkem.

První krok po dosažení konfigurace přenese přebytek nejvyššího vnitřního vrcholu do vrcholu za ním následujícího. Převedený přebytek se přičte k přebytku cílového vrcholu a v následujícím kroku se sloučené přebytky převádí dále společně. Vidíte, že tato varianta přebytky nezpracovává odděleně, ale slévá je dohromady a přenáší aglomerovaný přebytek v jedné operaci přenosu, čímž se dosáhne výrazné úspory výpočetního času.

Scéna: *Goldbergův algoritmus*

Závěrečná scéna umožní rekapitulaci všecho, co bylo vyloženo. Je možné volit různé příklady nebo si vytvořit svoji síť, vybrat si variantu algoritmu i délku kroku a sledovat výpočet. Pohyby myší také umožňují síť naklápět a rotovat.

10 Laboratoř

Goldbergův algoritmus pracuje s nehotovými toky a potřebuje mít kromě informace o rezervách hran pomocí proměnných $R(h)$ popisovaných v části o Ford-Fulkersonově algoritmu také informaci o přebytcích vnitřních vrcholů. Abychom je nemuseli stále pracně vypočítávat, použijeme pro každý vrchol v proměnnou $E(v)$, jejíž hodnoty upravujeme tak, aby byly stále rovny přebytkům příslušných vrcholů. Dosáhneme toho tím, že inicializaci algoritmu a přesun přebytku budeme provádět následujícím způsobem

for `každou hranu` h **do** $R(h) = c(h)$;
for `každý vrchol` v **do** $E(v) = 0$;

Inicializace

$R(h) = R(h) - \Delta$;
$R(h^{op}) = R(h^{op}) + \Delta$;
$E(u) = E(u) - \Delta$;
$E(v) = E(v) + \Delta$;

Přesun přebytku Δ přes hranu $h = (u, v)$

Pro zjednodušení popisu algoritmu zavedeme následující pojem: hrana $h = (u, v)$ se nazývá *spádová*, jestliže $H(u) > H(v)$, kde $H(u)$ a $H(v)$ jsou výšky vrcholů u a v. Velkou výhodou Goldbergova algoritmu je jeho jednoduchost:

```
Proměnné:  pro každý vrchol jeho výška H(v);
           pro každý vrchol číslo E(v), údaj o přebytku;
Inicializace;
for každou hranu h vycházející ze zdroje do
    přesuň přebytek c(h) přes hranu h;
while existuje vnitřní vrchol v takový, že E(v) > 0 do begin
    zvol libovolný vnitřní vrchol v takový, že E(v) > 0;
    if existuje spádová nenasycená hrana vycházející z v then begin
        zvol libovolnou spádovou nenasycenou hranu h vycházející z v;
        přesuň přebytek min(E(v), R(h)) přes hranu h;
        end
    else H(v) ← H(v) + 1;
    end;
```

Goldbergův algoritmus

11 Škola

Nyní dokážeme správnost a výpočetní složitost Goldbergova algoritmu.

Tvrzení 34 *Každý vrchol různý od zdroje má nezáporný přebytek.*

Důkaz: Zřejmý z popisu algoritmu (volba velikosti převáděného přebytku). ♣

Nyní ukážeme, že nenasycená hrana nemůže jít příliš "z kopce".

Tvrzení 35 *Během výpočtu Goldbergova algoritmu stále platí, že má-li hrana (v, w) kladnou rezervu, pak $H(v) \leq H(w) + 1$.*

Důkaz: Tvrzení dokážeme indukcí podle počtu opakování **while**-cyklu algoritmu. Je zřejmé, že po provedení inicializace platí, protože jediné hrany, jejichž výchozí a koncový vrchol jsou v různých výškách jsou hrany vycházející ze zdroje a ty mají nulovou rezervu.

Jsou dvě možnosti, jak by se mohlo tvrzení během výpočtu porušit. Při převodu přebytku zůstávají výšky nezměněny a proto by musela existovat hrana (v, w) taková, že $H(v) > H(w) + 1$ a její původně nulová rezerva se zvýší na nenulovou hodnotu. K tomu ale může dojít pouze tehdy, když je převáděn přebytek hranou opačnou, tedy hranou (w, v). Tato hrana ale jde "do kopce" a tedy v ní převádění přebytku není povoleno.

Při zvedání vrcholů naopak zůstávají nezměněny rezervy hran a mění se výšky. K porušení tvrzení lemmatu by tedy musela existovat hrana (v, w) s kladnou rezervou taková, že $H(v) = H(w) + 1$ a dojde ke zvednutí vrcholu v. Vrchol v by ale byl zvedán jen v případě, kdy vrchol v má kladný přebytek, ale v takové situaci algoritmus přikazuje převést přebytek z vrcholu v hranou (v, w) a nikoli zvedání vrcholu v. ♣

Toto jednoduché tvrzení nám již umožní dokázat správnost výpočtu v případě jeho ukončení.

Tvrzení 36 *Pokud se Goldbergův algoritmus zastaví, nalezl největší tok.*

Důkaz: K zastavení dojde v okamžiku, kdy přebytky vrcholů jiných než zdroj a spotřebič jsou nulové, konstruovaný nehotový tok je tedy tok. Je-li $z = v_0, v_1, \ldots, v_k = s$ libovolná prostá cesta ze zdroje do spotřebiče a označíme-li si jako n počet vrcholů sítě, má cesta nejvýše $n - 1$ vrcholů (tedy $k < n$), ale překonává výškový spád n, takže na ní bude ležet hrana s rozdílem výšky počátku a konce alespoň 2 a ta podle předchozího lemmatu má nulovou rezervu. Z toho již plyne, že je tok maximální, viz kapitola o Ford-Fulkersonově algoritmu, Věta 10. ♣

Poznamenejme, že z tohoto lemmatu je vidět, proč je výška zdroje volena n. Pokud by výška byla menší, mohlo by dojít k zastavení v okamžiku, kdy nalezený tok ještě není největší možný. Jak bude ukázáno dále, pokud by výška zdroje byla větší, výpočet by sice proběhl správně, ale trval by déle.

Tvrzení 37 *Jestliže kdykoliv během výpočtu Goldbergova algoritmu má vrchol v kladný přebytek, pak existuje prostá orientovaná cesta z v do zdroje taková, že každá její hrana má kladnou rezervu.*

Důkaz: Nechť v jistém okamžiku má v kladný přebytek. Označme si jako A množinu vrcholů takovou, že $w \in A$ právě tehdy, když existuje prostá cesta z v do w taková, že každá její hrana má kladnou rezervu. Naším cílem je tedy dokázat, že zdroj patří do A.

Nechť (x, y) je hrana taková, že $x \notin A$ a $y \in A$. Pokud by tok hranou (x, y) byl nenulový, hrana (y, x) by měla kladnou rezervu. Prodloužení prosté cesty z v do y složené z hran s kladnými rezervami o hranu (y, x) by dalo prostou cestu z v do x s kladnými rezervami hran, což by bylo ve sporu s předpokledem, že x neleží v A. Toky hranami vstupujícími do A z vrcholů mimo A jsou tedy nulové. Nyní platí

$$\sum_{w \in A} exc(w) = \sum_{w \in A} \left[\sum_{h \in in(w)} t(h) - \sum_{h \in out(w)} t(h) \right] = \sum_{w \in A} \sum_{h \in in(w)} t(h) - \sum_{w \in A} \sum_{h \in out(w)} t(h).$$

Hrana h, která má oba konce mimo A, v sumách ve výrazu na pravé straně nevystupuje, zatímco hrana, která má oba konce v A vystupuje v obou sumách, tedy jednou se $t(h)$ přičítá a jednou odečítá a celkový příspěvek je nulový. Hrany, které začínají mimo A a končí v A mají, jak bylo ukázáno výše, nulový tok, takže

$$\sum_{w \in A} exc(w) = - \sum_{h \in out(A)} t(h) \leq 0.$$

Součet přebytků vrcholů ležících v A je tedy menší nebo roven nule, a jelikož vrchol v je v A a má kladný přebytek, musí v A ležet i vrchol se záporným přebytkem, ale takový je pouze jeden - zdroj. ♣

Toto lemma má velmi důležitý důsledek:

Tvrzení 38 *Výška žádného vrcholu v průběhu výpočtu Goldbergova algoritmu není větší než $2|V|$.*

Důkaz: Označme $n = |V|$. Kdyby lemma neplatilo, musel by v průběhu výpočtu nastat okamžik, kdy některý vrchol v má výšku $2n$ a je zvedán. Jelikož je v zvedán, musí mít kladný přebytek, viz algoritmus. Pak ale existuje prostá cesta z v do zdroje, tvořená hranami s kladnou rezervou. Jelikož je tato cesta prostá, obsahuje nejvýše $n - 1$ hran a přitom překonává výšku n, protože zdroj je ve výšce n. Pak alespoň jedna hrana této cesty překonává spád větší než 1, ale jelikož má kladnou rezervu, dostáváme tím spor s lemmatem 35. ♣

Z tohoto lemmatu také plyne odhad pro počet zvednutí:

Tvrzení 39 *V průběhu celého výpočtu se provede operace zvednutí nejvýše $2n^2$ krát.*

Důkaz: Vrcholů je n, každý (s výjimkou zdroje, který není zvedán) začíná ve výšce 0 a končí, viz předchozí lemma, nejvýše ve výšce $2n$ a jeho výška se může jen zvětšovat a nikdy neklesá, takže jednotlivý vrchol může být zvedán nejvýše $2n$ krát. ♣

Nyní odhadneme počet operací převedení přebytku. Odhad provedeme odděleně pro nasycené a nenasycené převody. Jednodušší je odhad nasycených převodů, který je také nižší.

Tvrzení 40 *V průběhu celého výpočtu se provede operace nasyceného převedení přebytku nejvýše nm krát, kde n je počet vrcholů a m je počet hran sítě.*

Důkaz: Uvažujme hranu (u, v). Nasycené převedení přebytku touto hranou může nastat pokud $H(u) \geq H(v) + 1$ (viz popis algoritmu a Lemma 35) a způsobí, že její rezerva poklesne na nulu, takže k případnému novému převádění přebytku může dojít až poté, co se rezerva hrany zase zvýší na kladnou hodnotu. K tomu může dojít pouze když je převáděn přebytek opačnou hranou (v, u) a to může nastat pouze tehdy, když platí $H(v) \geq H(u) + 1$. Jelikož $H(u)$ nemůže klesnout, musí se tedy $H(v)$ zvýšit nejméně o 2.

V okamžiku, kdy rezerva hrany (u, v) se z nuly zvýšila na nenulovou hodnotu, je tedy $H(v) \geq H(u) + 1$, ale k novému převedení přebytku hranou (u, v) může dojít pouze když $H(u) \geq H(v) + 1$ a jelikož $H(v)$ nemůže klesnout, musí se $H(u)$ zvýšit alespoň o 2.

Celkově se tedy mezi dvěma po sobě následujícími nasycenými převedeními přebytku hranou (u, v) musí jak $H(u)$, tak i $H(v)$ zvýšit alespoň o 2, tedy součet $H(u) + H(v)$ vzroste o nejméně 4.

Součet $H(u) + H(v)$ je minimálně 0, může se jen zvětšovat a na základě předchozího lemmatu je nejvýše $4n$. Z toho plyne, že jednou hranou lze nasyceně převádět přebytek nejvýše n krát. Hran je přitom m. ♣

Nakonec nejsložitější je odhad počtu nenasycených převodů přebytku, protože při této operaci rezerva hrany neklesne na nulu a proto je ji možno provádět opakovaně bez změny výšek vrcholů. Viděli jsme ve scéně s volbou nejnižšího vrcholu s přebytkem, že počet nenasycených převodů může být značný.

Tvrzení 41 *V průběhu celého výpočtu se provede operace nasyceného převedení přebytku nejvýše $(m + 2)n^2$ krát, kde n je počet vrcholů a m je počet hran sítě.*

Důkaz: Definujme

$$\Gamma = \sum_{\substack{exc(v) > 0 \\ z \neq v \neq s}} H(v),$$

neboli Γ je součet výšek vnitřních vrcholů sítě s kladnými přebytky. Podívejme se nyní, jak hodnotu Γ mění jednotlivé operace.

Operace zvednutí vrcholu zvýší výšku vrcholu s kladným přebytkem o 1 a tedy také zvýší hodnotu Γ o 1.

Operace nasyceného převedení přebytku hranou (u, v) sníží přebytek u a zvýší přebytek v, takže se může stát že ze součtu vypadne člen $H(u)$ (pokud by přebytek u klesl na nulu) a že do součtu přibyde $H(v)$ (pokud přebytek v byl před operací nulový). Celkově tedy se Γ nemůže zvýšit o více než $H(v) \leq 2n$.

Operace nenasyceného převedení přebytku hranou (u, v) vždy sníží přebytek u na nulu a tedy člen $H(u)$ ze součtu pro Γ vypadne. To může být kompenzováno tím, že do součtu přibyde $H(v)$, ale v okamžiku provádění operace musí být $H(u) - 1 \geq H(v)$, takže v každém případě se nenasyceným převedením přebytku sníží Γ o alespoň 1, neboli přírustek Γ je nejvýše -1.

Hodnota Γ po provedení inicializace je 0 (všechny vnitřní vrcholy mají výšku 0) a na konci výpočtu také 0 (žádný vnitřní vrchol nemá kladný přebytek). Označíme-li jako Δ součet přírustků hodnoty Γ pro jednotlivé operace, musí být $\Delta \geq 0$.

Označíme-li ale jako k počet provedení operace nenasyceného převedení přebytku hranou, pak z odhadů pro počty provedení operací zvednutí vrcholu a nasycených převedení a z odhadů pro přírustky Γ při jednotlivých operacích dostáváme

$$0 \leq \Delta \leq 1 \cdot 2n^2 + 2n \cdot mn + (-1) \cdot k = 2n^2 + mn^2 - k$$

a z této nerovnosti vyplývá okamžitě $k \leq 2n^2 + mn^2$. ♣

Shrnutím tedy dostáváme

Věta 42 *Goldbergův algoritmus zpracuje síť s n vrcholy a m hranami po $O(mn^2)$ krocích.*