Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Toky v sieťach

(bakalárska práca)

Richard Štefanec

vedúci:
Mgr.Michal Forišek

Bratislava, 2007

Čestne prehlasujem, že predkladanú bakalársku prácu som vypracoval samostatne s použitím uvedenej literatúry.

.....

Abstrakt

Problémy z rôznych oblastí sa dajú preformulovať na úlohy hladania maximálneho toku v sieti. Existuje viacero spôsobov, ako k ich riešeniu pristupovať – najčastejšími metódami sú modifikácie algoritmu Forda a Fulkersona a metóda preflow-push. Implementovali sme niekoľko algoritmov založených na týchto metódach a niekoľko generátorov grafov. Experimentálne sme zisťovali, ktoré z algoritmov je výhodné použiť pri riešení úloh na grafoch z rôznych tried.

Kľúčové slová: toky v sieťach, maximálny tok, Ford-Fulkerson, grafové algoritmy

Obsah

1 Úvod **1**

2 Prehľad **3**

3 Teoretické základy **7**

4 Tokové algoritmy **11**
4.1 Metóda zlepšujúcich ciest 11
4.2 Preflow-push 13
4.3 Implementácia 17
4.4 Redukcie 19

5 Špeciálne typy grafov **23**
5.1 Siete malého sveta 23
5.2 Regulárne grafy 23
5.3 Bipartitné grafy 24
5.4 Riedke a husté grafy 24
5.5 Iné triedy grafov 25

6 Praktické testy **27**

7 Záver **35**

Literatúra **41**

Kapitola 1

Úvod

Mnoho úloh z rôznych oblastí sa dá previesť na úlohu hľadania maximálneho toku. Zároveň často druh problému priamo určuje typ grafu, na ktorom sa úloha o maximálnom toku má riešiť. Preto je zmysluplné skúmať, ktorý zo známych algoritmov je vhodný na použitie pre daný typ grafu a ktorý nie. V tejto práci sme sa pokúsili zaviesť základné pojmy z oblasti tokov v sieťach, ukázať a implementovať niekoľko základných algoritmov, ktoré sa pri riešení tejto úlohy používajú a pokúsiť sa na základe empirického skúmania zistiť, ako sa správajú na jednotlivých typoch grafov. Zamerali sme sa na algoritmy z dvoch najrozšírenejších metód – Preflow-push a metódy hľadania zlepšujúcich ciest.Zaujímali sme sa nielen o čas ich behu, ale i o to, ako sa v rámci jednej triedy algoritmov líšia v množstve volaní jednotlivých procedúr na rovnakých vstupoch. Dôvod pre skúmanie času behu týchto algoritmov v praxi je veľký rozdiel medzi maximálnou a priemernou zložitosťou.

Pri tvorbe tejto práce sme vychádzali predovšetkým z knihy Graph Algorithms in C++ od R.Sedgewicka [4] a Network Flows (Ahuja, Magnanti, Orlin) [1]. Jednotlivé algoritmy sme implementovali v programovaciom jazyku C++. Sú kompilovateľné pomocou kompilátora gcc a nezávislé na platforme. V prípade záujmu o danú problematiku môžete v [4] nájsť pomerne neformálny, ale vyčerpávajúci úvod, monografia [1] okrem iného ukazuje veľké množstvo praktických problémov, ktoré sa pomocou jednotlivých tokových algoritmov dajú riešiť.

Kapitola 2

Prehľad

Tok ako matematický pojem je zovšeobecnením situácie, kedy potrebujeme z nejakého bodu cez cestnú sieť dostať isté množstvo tovarov do iného bodu, prípadne fyzikálnej reality toku tekutín alebo elektriny. V našej práci sa budeme venovať úlohe o maximálnom toku, v ktorej ide o to maximalizovať množstvo materiálu prechádzajúceho sieťou, ktorá má na jednotlivých hranách množstevné obmedzenia.

Existuje veľké množstvo úloh, ktoré sa dajú riešiť pomocou tokov, od úplne prirodzených úloh o presune materiálu až po rôzne zaujímave kombinatorické úlohy, kde práve oblasť tokov v sieťah ukazuje zaujímavé súvislosti. V tejto kapitole sa pokúsime ukázať niekoľko oblastí, kde výraznou časťou riešenia problému je práve vyriešenie úlohy o hľadaní maximálneho toku. Tieto sa dajú zaradiť do troch základných kategórií:

- Distribučné problémy (distribution)
- Párovanie (matching)
- Rezy (cuts)

Pri riešení distribučných problémov sa zaoberáme presúvaním objektov z jedného miesta na druhé. Najčastejšími aplikáciami je presun tovaru, kde máme niekoľko výrobných závodov, dopravnú infraštruktúru, kde po jednotlivých trasách sme schopní dostať niekoľko jednotiek tovaru za jednu jednotku času a miesta kam potrebujeme daný tovar presunúť. Otázka môže byť položená tak, či sme schopní presunúť isté množstvo tovaru, alebo aké je maximálne množstvo, ktoré sme za jednu jednotku času schopní dodávať.

Inou zaujímavou aplikáciou je tok dopravy, kedy nás môže zaujímať, koľko vozidiel sme schopní po existujúcej infraštruktúre dostať z istej oblasti za nejaké časové obdobie v prípade, že by sme vedeli riadiť dopravu pomocou rôznych presmerovaní a nariadení, prípadne kadiaľ by sme túto dopravu mali usmernovať aby sme maximalizovali počet vozidiel opúšťajúcich oblasť. Prípadne nás môže zaujímať, kde upraviť infraštruktúru tak, aby sme čo najefektívnejšie zvýraznili priepustnosť, alebo kde sú miesta, v ktorých v prípade zvýšeného počtu vozidiel bude dochádzať k zápcham.

Tieto úlohy sa dajú abstrahovať tak, že sa na infraštruktúru pozeráme ako na graf, kde cesty sú hrany, ktoré majú istú priepustnosť, ktorú označíme kapacita. Križovatky, prekladiská, sklady, alebo predajne sú vrcholy. Vrcholy v ktorých čosi vzniká označíme ako zásobovacie (supply), vrcholy kam potrebujeme tovar dostať ako dopytové (demand) a všetky ostatné vrcholy v sieti ako distribučné (distribution).

Pri párovaní máme dve množiny objektov, ktoré sú medzi sebou prepojené podľa istých pravidiel. Našou úlohou je vybrať spomedzi týchto konexií maximálne množstvo, ak má byť dodržaná podmienka, že každý objekt je prepojený najviac s jedným iným objektom. V reálnom živote môže ísť napríklad o hladanie maximálneho množstva tanečných párov v skupine mužov a žien, kde vieme o jednotlivých potenciálnych dvojiciach, či spolu chcú alebo nechcú tancovať. Prípadne môžeme mať množinu ľudí, kde každý je schopný vykonávať rôzne typy činností a počet jednotlivých voľných miest rôznych druhov. Našim záujmom je nájsť každému členovi činnosť, na ktorú je kvalifikovaný, alebo aspoň maximalizovať počet členov, ktorých vieme zamestnať.

Úlohy týkajúce sa párovania na prvý pohľad nepôsobia tak, že by nám pri ich riešení toky mohli nejak pomôcť, avšak opak je pravdou. Stačí keď budeme jednu skupinu objektov považovať za zásobovacie vrcholy, druhú za dopytové a jednotlivé prepojenia budú hranami s kapacitou 1. Tento graf si upravíme tak, aby z každého zásobovacieho vrcholu vytekala práve jedna jednotka a každý dopytový bol schopný prijať tiež len jednu jednotku (kvôli tomu aby sme nijaký objekt nemohli spáriť s dvoma rôznymi). To dosiahneme napríklad tak, že pripojíme ku grafu jeden vrchol, ktorý nazveme zdroj a z ktorého bude vychádzať do každého zásobovacieho vrcholu hrana s kapacitou 1 a podobne z každého dopytového vytvoríme jednotkovú hranu smerujúcu do vrcholu ktorý nazveme ústím grafu. Teraz hľadanie maximálneho množstva tovaru ktorý sme schopný prepraviť medzi zdrojom a ústim je zároveň hľadaním maximálneho množstva párov ktoré sme schopní prepojiť.

Tretia oblasť, kde nám môžu toky pomôcť je riešenie úloh o rezoch. Tieto úlohy sa často objavujú pri príprave infraštruktúry, alebo pri hľadaní slabých miest v komunikačných sieťach. Ukážeme si dve modelové situácie.

Majme cestnú sieť pomocou ktorej prepravujeme zásoby spoza bojovej línie na front. Každá cesta v tejto sieti má trocha iné vlastnosti a teda je pre nepriateľa viac či menej zložité ju zničiť tak aby sme po nej neboli schopný zásoby prepravovať. Nech vieme koľko bômb by nepriateľ potreboval na znefunkčnenie jednotlivých úsekov ciest, tak nás môže zaujímať aké je minimálne potrebné množstvo bômb na to aby nám uplne znemožnil dopravu do prvej línie. Naopak nepriateľa môže zaujímať, na ktoré úseky sa zamerať aby minimalizoval množstvo použitej munície.

Iná úloha ktorá sa dá vyriešiť pomocou nájdenia minimálneho rezu je v oblasti telekomunikácií. Nech spravujeme telekomunikačnú sieť, môže nás zaujímať, koľko úsekov tejto siete môže byť v jednom momente mimo prevádzky bez toho, aby to ovplyvnilo spojenie medzi bodom a a b.

Ako si neskôr (veta 4.1) ukážeme, hľadanie maximálneho toku a minimálneho rezu vzájomne úzko súvisia, takže na vyriešenie predchádzajúcich úloh o rezoch bude stačiť vyriešiť problém maximálneho toku.

Mimo týchto oblastí sa maximálne toky objavujú ako podproblém pri riešení zložitejších tokových úloh ako sú viackomoditné toky, alebo úlohy o maximálnom toku s minimalnou cenou (mincost flow problem), preto efektívne riešenie tejto úlohy môže mať výrazný vplyv na riešenie zložitejších úloh.

Kapitola 3

Teoretické základy

Definícia 3.1. Orientovaný graf G *je dvojica* (V, U)*, kde* V *je množina vrcholov a* E *je konečná množina usporiadaných dvojíc vrcholov z* V*, ktoré budeme označovať pojmom hrana. Ďalej predpokladáme, že* E *neobsahuje nijakú hranu* (v, v)*, kde* $v \in V$*, tj. graf neobsahuje slučky. Podobne množina hrán obsahuje najviac jednu dvojicu* $(u, v) \in E$*, kde* $u, v \in V$*, teda graf nemá násobné hrany.*

U neorientovaného grafu *pozostáva množina* E *z neusporiadaných dvojíc vrcholov. Držiac sa zaužívaných konvencií jej prvky nebudeme označovať* $\{u, v\}$*, ale pomocou pojmov* (u, v) *resp.* (v, u) *ktoré budú oboje označením pre jednu a tú istú hranu.*

Orientovaný graf budeme nazývať aj digraf, oboje následne pojmom graf, ak bude z kontextu jesné o ktorý z nich sa jedná.

Mohutnosť množiny V *grafu* G *budeme označovať* N_G*, prípadne* N *na miestach kde bude jasné o aký graf* G *sa jedná, podobne mohutnosť množiny* E *budeme označovať* M_G *resp.* M*.*

Definícia 3.2. *Orientovaný graf* $G = (V, E)$*, kde každej hrane* $(u, v) \in E$ *je priradená nezáporná* kapacita $c(u, v) \geq 0$ *nazývame* st-sieť*. Ak* $(u, v) \notin E$*, predpokladáme, že* $c(u, v) = 0$*, naopak niekedy budeme hrany* s $c(u, v) = 0$ *označovať ako neexistujúce. Rozlišujeme dva špeciálne vrcholy –* zdroj s *(source) a* ústie t *(sink), kde* $s \neq t$*. Pre všetky vrcholy z* V *predpokladáme, že ležia na nejakej ceste z* s *do* t*.*

V prípade, že pre všetky $(u, v) \in E$ *platí* $c(u, v) = 1$*, tak hovoríme že sieť má jednotkovú kapacitu hrán.*

Definícia 3.3. Tok *je funkcia* $f : V \times V \to \mathbb{N}$ *pre ktorú platia nasledujúce dve podmienky:*

(a) $0 \leq f(u, v) \leq c(u, v)$ *pre všetky* $u, v \in V$

(b) $\sum_{u \in V} f(u, v) = \sum_{w \in V} f(v, w)$ *pre všetky vrcholy* $v \in V - \{s, t\}$

Sumu $\text{outflow}(v) = \sum_{u \in V} f(v, u)$ *budeme nazývať* odtok *(outflow) z vrcholu* v *a* $\text{inflow}(v) = \sum_{w \in V} f(w, v)$ *budeme nazývať* prítok *(inflow) do vrcholu* v.

Pod hodnotou toku budeme rozumieť odtok z vrcholu s.

Maximálny tok *je taký, pre ktorý neexistuje iný tok v danej sieti s väčšou hodnotou.*

Poznámka 3.1. *Obmedzenie sa na grafy bez násobných hrán a slučiek nespôsobuje nijaký problém a väčšina tvrdení uvedených v tomto texte by ostala v platnosti i v prípade, že by sme pracovali s grafmi, ktoré slučky a násobné hrany majú. Ľahko vidieť, že ak by sme mali miesto jednej hrany niekoľko násobných, tak tieto vieme reprezentovať jednou hranou, ktorej kapacita je rovná súčtu kapacít násobných a že slučky na ľubovoľnom vrchole nám nemenia hodnotu toku.*

Veta 3.1. *Pre všetky st-toky platí* $\sum_{u \in V} f(s, u) = \sum_{w \in V} f(w, t)$.

Dôkaz.
Dôkaz je možné nájsť v [4] ako tvrdenie 22.1. □

Táto veta hovorí o tom, že vrámci siete sa tok nemôže stratiť. V prípade, že by sme prepojili ústie so zdrojom, tak by nám vznikla cirkulácia, kde platí lokálne ekvilibrium pre všetky vrcholy, vrátane s a t a hodnota toku v celej sieti je práve hodnota toku na hrane medzi ústim a zdrojom.

Definícia 3.4. Rez *(cut) st-siete (S,T) je rozdelenie množiny vrcholov na dve časti – S a $T = V - S$, také, že $s \in S$ a $t \in T$. Každý rez jednoznačne určuje množinu hrán ktoré majú jeden koniec v S a druhý v T, ktorú rovnako označujeme rez. Kapacitou rezu označujeme súčet kapacít hrán $(u, v) \in E$ takých, že $u \in S$ a $v \in T$. Označujeme ju $c(S, T)$.*

Minimálny rez *je rez s najnižšou kapacitou v st-sieti.*

Teraz si zavedieme dva základné pojmy nutné pre pochopenie tokov v sieťach – pojem reziduálnej siete a zlepšujúcej cesty.

Majme st-sieť s ľubovoľným tokom. Reziduálna sieť k danej st-sieti nám vlastne hovorí o tom, cez ktoré hrany sa dá poslať nenulové množstvo toku. Majme (orientovanú) hranu $(u, v) \in E$ s kapacitou 5 a tokom 2, zaujíma nás o koľko vieme zvýšiť tok z bodu u tak, aby bola dodržaná podmienka $f(u, v) \leq c(u, v)$, prirodzene v tomto prípade to je 3, tj. $c(u, v) - f(u, v)$, opačným smerom sme po hrane taktiež schopný poslať tok, čo v konečnom dôsledku spôsobí to, že reálne po danej hrane pošleme o toľko jednotiek menej. Maximum, ktoré sme schopní takto prepraviť proti orientácií hrany je $f(u, v)$. Každej hrane v st-sieti priradíme dve hrany s opačnou orientáciou. Každá z nich označuje množstvo toku ktoré sme schopní pridať k (u, v) v danom smere, tak, aby nebola porušená podmienka 3.3(a). Zavedieme si to formálne.

Definícia 3.5. *Nech $G = (V, E)$ je st-sieť a f je tok,* reziduálna sieť *(residual network) k tejto sieti a toku je $G = (V, E')$, kde pre každú hranu $e = (u, v) \in E$ s kapacitou c a hodnotou toku f priradíme v reziduálnej sieti dve hrany, jednu s orientáciou (u, v) a kapacitou $c - f$ a druhú s orientáciou (v, u) a kapacitou f.*

Pojem reziduálnej siete je úzko previazaný s kapacitou a aktuálnymi hodnotami toku v sieti. Ako vidno z definície, ku každej st-sieti vieme jednoznačne priradiť reziduálnu sieť a naopak, reziduálna sieť nám jednoznačne určuje hodnoty toku na jednotlivých hranách v st-sieti. Preto možeme namiesto práce s st-sieťou pracovať s jej reziduálnou sieťou a po nájdení maximálneho toku tento tok vyjadriť priamo z reziduálnej siete.

Druhý pojem, ktorý je potrebné zaviesť je pojem zlepšujúcej cesty.

Definícia 3.6. Zlepšujúca cesta *(augmenting path) je cesta v reziduálnej sieti z vrcholu s do vrcholu t kde každá hrana na tejto ceste má kladnú kapacitu.*

Je nutné si uvedomiť, že hrany na tejto ceste môžu viesť aj proti svojej orientácií v pôvodnej st-sieti. Množstvo toku ktorý sme schopní poslať po tejto ceste je rovné minimu z kapacít jej hrán.

Kapitola 4

Tokové algoritmy

Pri riešení úloh o hľadaní maximálneho toku sa ukázali ako užitočné dva rôzne prístupy k tomuto problému, z ktorých každý indukuje celú triedu algoritmov. V tejto kapitole si najskôr oba zavedieme, ukážeme ako sme jednotlivé algoritmy z týchto tried implementovali a nakoniec ukážeme, že viaceré úlohy vieme preformulovať na úlohy o hladaní maximálneho toku, tak ako sme ju zaviedli.

4.1 Metóda zlepšujúcich ciest

Táto metódu prvýkrát uviedli L. R. Ford a D. R. Fulkerson v roku 1962. V literatúre sa zvykne nazývať rôzne, my ju budeme označovať ako *metódu zlepšujúcich ciest*, alebo *Ford-Fulkersonovu metódu* (Ford-Fulkerson method, augmenting-path maxflow method).

Ide o iteratívnu metódu, začíname s $f(u, v) = 0$ pre všetky $u, v \in V$. Pri každej iterácií zvýšime hodnotu toku nájdením zlepšujúcej cesty (def. 3.6) a poslaním maximálneho možného toku po tejto ceste. Ako si neskôr ukážeme, tento proces vyústi do nájdenia maximálneho toku.

Priebeh hľadania maximálneho toku, sa dá zapísať nasledujúcim pseudokódom:

GenericFordFulkerson(G, s, t)

```
1  inicializuj tok na všetkých hranách na 0
2  while existuje zlepšujúca cesta p
3        do pošli tok po ceste p
```

Veta 4.1. *(Maxflow-mincut theorem) Hodnota maximálneho toku v st-sieti je ekvivalentná hodnote minimálnej kapacity spomedzi všetkých st-rezov.*

Dôkaz.
Dôkaz je možné nájsť napríklad v [4] ako tvrdenie 22.5, pre naše potreby je dôležitá len myšlienka za jednou z inklúzií. Nech prebehne Ford-Fulkersonov algoritmus a už nie je možné nájsť nijakú s-t cestu. Preto sa pozrieme na všetky vrcholy, do ktorých existuje zlepšujúca cesta. Zjavne ak rozdelíme graf na dosiahnutelné vrcholy (kam patrí s) a nedosiahnuteľné (kam patrí t), tak hrany medzi nimi tvoria st-rez. Navyše všetky st hrany sú saturované a ts hrany majú nulový tok. Teda hodnota tohto rezu je rovnaká ako hodnota toku. □

Veta 4.2. *(korektnost augumenting paths algoritmu) Ford-Fulkersonov algoritmus vypočíta maximálny tok.*

Dôkaz.
Najskôr potrebujeme ukázať, že algoritmus ukončí činnosť v konečnom čase. Ľubovoľný tok má konečne veľkú hodnotu, keďže kapacity hrán sú konečné, takže jeho hodnotu vieme obmedziť napríklad rezom $(\{s\}, V - \{s\})$. Každá zlepšujúca cesta tento tok zvýši minimálne o 1. Preto po konečnom počte iterácií nájdeme nejaký tok a nevieme nájsť ďalšiu zlepšujúcu cestu. Že algoritmus vypočíta maximálny tok priamo vyplýva z vety 4.1. Táto nám navyše dáva návod ako nájsť jeden z minimálnych rezov. □

Generická metóda nám neurčuje spôsob, akým máme hľadať zlepšujúce cesty, zato nám však ukazuje, že akýmkoľvek spôsobom ich budeme vyberať, tak to neovplyvní správnosť algoritmu. Práve spôsob akým sa tieto cesty hľadajú výrazne určuje efektívnosť algoritmov, rovnako ako i ďalšie parametre nájdeného maximálneho toku ako napríklad počet hrán s nenulovou hodnotou toku.

My sme implementovali tri základné spôsoby prehľadávania grafu – prehľadávanie do šírky, ktoré nachádza najkratšiu cestu zo zdroja do ústia, prehľadávanie do hĺbky a hľadanie cesty po ktorej je možné nechať tiecť najväčšie množstvo toku.

Bližšie sa týmto prístupom budeme venovať v kapitole o implementácií.

4.2 Preflow-push

Preflow-push pracuje omnoho lokálnejšie, než Ford-Fulkersonova metóda. Namiesto spracúvania celej siete pri hľadaní zlepšujúcich ciest v tejto metóde pracujeme v každom momente len na jednom vrchole a vrcholoch s ním susediacich v reziduálnej sieti. Ďalší rozdiel je, že počas svojho behu nespĺňa podmienku $\sum_{u \in V} f(u,v) = \sum_{w \in V} f(v,w)$ pre $v \in V - \{s,t\}$ z definície 3.3. Spĺňa však menej prísnu podmienku, ktorú budeme označovať preflow.

Definícia 4.1. *V st-sieti,* preflow *je funkcia* $f : V \times V \to \mathbb{N}$ *pre ktorú platí, že* $0 \leq f(u,v) \leq c(u,v)$ *pre všetky* $u, v \in V$ *a pre všetky vrcholy* $v \in V - \{s\}$ *platí* $\sum_{u \in V} f(u,v) \geq \sum_{w \in V} f(v,w)$. *Vrcholy pre ktoré platí nerovnosť* $\sum_{u \in V} f(u,v) > \sum_{w \in V} f(v,w)$ *označujeme ako* aktívne. *Vrcholy s a t podla konvencie nemôžu byť počas behu aktívne.*

Priebeh preflow-push metódy si môžeme predstaviť nasledovne. Nech graf G je systém neobmedzene veľkých zásobníkov prepojených rúrami. Nech tieto zásobníky sú na začiatku vykonávania v rôznych výškach a nech ústie je vo výške 0. Najskôr zo zdroja vypustíme maximálne množstvo vody do zásobníkov jeho susedov. Počas behu si postupne vyberáme aktívne vrcholy a buď na nich vykonáme operáciu push, ktorá pošle z daného zásobníka maximálne množstvo vody do niektorého z nižšie postavených susedov, alebo ak taký sused nie je, tak zvýšime výšku zásobníka. Vo chvíli keď dostaneme do ústia všetkú vodu, ktorú nám tam daný systém potrubí dovolí dostať, tak nám v zásobníkoch môže zostať zbytková voda, ktorú postupne pošleme naspäť do zdroja.

K formálnemu zadefinovaniu potrebujeme presne definovať výškovú funkciu.

Definícia 4.2. Výšková funkcia *(height function) pre daný tok v st-sieti je funkcia* $h : V \to \mathbb{N}$ *pre ktorú platí,* $h(t) = 0$ *a* $h(u) \leq h(v) + 1$ *pre všetky hrany* (u,v) *z reziduálnej siete. Hrana* (u,v) *sa označuje* vhodná *ak pre ňu platí podmienka* $h(u) = h(v) + 1$.

Preflow-push metóda pracuje podľa nasledujúceho pseudokódu. Po zavolaní PreflowPush najskôr vykoná prípravu grafu, kde vynuluje tok, v prípade, že je nenulový, vypočíta výšky jednotlivých vrcholov podľa danej výškovej funkcie tak aby spĺňali definíciu 4.2, potom pošle maximalné množstvo toku z vrcholu s do jeho susedov, čo je i poslednýkrát kedy z vrcholu s

nechávame tiecť nejaký tok. Následne nastaví výšku vrcholu s na $|N$, aby sa nestalo, že do s pritečie nejaký tok v bode, keď budú existovať ešte nejaké inéy možnosti jeho susedov ako poslať tok smerom k ústiu. (toto zvýšenie nám neporuší podmienky z výškovej funkcie, lebo v danom momente v reziduálnej sieti neexistuje nijaká kladná hrana so začiatkom vo vrchole s)

Ďalej funkcia PreflowPush zisťuje či existuje nejaký aktívny vrchol a pokial existuje, tak niektorý vyberie a volá na ňom funkciu PushRelabel, ktorá sa z neho najskôr pokúša dostať tok do všetkých vhodných hrán a keď v tomto vrchole i potom ostáva prebytočný tok, tak zvýši jeho výšku (záleží od implementácie či ju len jednoducho inkrementuje, alebo či si bude pamätať najnižšiu výšku spomedzi svojich susedov a zvýši tok tak aby tam po vykonaní tejto funkcie existovala aspoň jedna vhodná hrana).

Preprocess(G, s, t)

1 vypočítaj hodnoty výškovej funkcie
2 **for** $j \leftarrow 0$ **to** počet susedov vrcholu s
3 **do** pošli maximálny možný tok po hrane (s, j)
4 nastav výšku vo vrchole s na N

PushRelabel(G, s, t, v)

1 **for** $j \leftarrow 0$ **to** počet susedov vrcholu v
2 **do if** hrana (v, j) je vhodná
3 **then** pošli maximum z hodnôt outflow(v) a $c_R(v, j)$ po hrane (v, j)
4 **if** outflow(v) − inflow$(v) > 0$
5 **then** zvýš výšku vrcholu v

PreflowPush(G, s, t)

1 Preprocess(G,s,t)
2 **while** existuje aktívny vrchol
3 **do** vyber aktívny vrchol v
4 PushRelabel(G,s,t,v)

Správnosť preflow-push metódy

Veta 4.3. *Preflow-push algoritmus zachováva platnosť výškovej funkcie.*

Dôkaz.
Po vykonaní procedúry Preprocess je výšková funkcia platná. Porušiť funkciu môžeme pri zvyšovaní výšky nejakého vrchola. Nech je to vrchol u. Tento

vrchol zvyšujeme len v prípade, že neexistuje hrana vedúca do nijakého vrchola s výškou h(u)-1. Keďže pred zvýšením bola výšková funkcia platná z indukčného predpokladu, tak musí platiť nerovnosť $h(u) < h(v) + 1$. Potom ale po zvýšení platí $h(u) \leq h(v) + 1$. (Úplne zjavne táto nerovnosť platí i v prípade, že zvyšujeme na hodnotu najnižšieho suseda + 1.) Zvýšenie $h(u)$ nijak neovplyvní nerovnosti tvaru $h(w) \leq h(u) + 1$, ani nerovnosti v ktorých vrchol u nevystupuje. Keďže vrcholy t a s z definície nikdy nie sú aktívne, tak tieto nikdy neinkrementujemne. Z týchto pozorovaní už vyplýva platnosť tvrdenia. □

Veta 4.4. *Pre ľubovoľný tok a výškovú funkciu platí, že výška ľubovoľného vrcholu v nie je vyššia ako dĺžka najkratšiej $v - t$ cesty v reziduálnej sieti.*

Dôkaz.
Nech pre v je najkratšia $v - t$ cesta $v = v_1 v_2 ... v_{n+1} = t$ dĺžky n, potom podľa definície 4.2 platí $h(v_i) \leq h(v_{i+1}) + 1$, teda $h(v_1) \leq h(v_{n+1}) + n$, kde $h(v_1) = h(v)$ a $h(t) = h(v_{n+1}) = 0$, teda $h(v) \leq n$ □

Veta 4.5. *Ak je výška vrcholu v vyššia alebo rovná ako N, tak v reziduálnej sieti neexistuje $v - t$ cesta.*

Dôkaz.
Ak je výška vrcholu v aspoň N, tak z vety 4.4 vyplýva, že dĺžka cesty medzi vrcholom v a ústim grafu je aspoň N, čo pri N vrcholoch nie je možné. □

Veta 4.6. *Počas behu preflow-push algoritmu na st-sieti, existuje v reziduálnej sieti cesta (s kladným ohodnotením hrán) z ľubovoľného aktívneho vrcholu do zdroja a neexistuje nijaká cesta zo zdroja do ústia v danej reziduálnej sieti.*

Dôkaz.
Indukciou: Po inicializácií sú aktívne len vrcholy ktoré susedia so zdrojom a hrany medzi zdrojom a týmito vrcholmi sú saturované, teda v reziduálnej sieti existuje nenulová hrana z každého aktívneho vrchola do zdroja a naopak zo zdroja neexistuje nijaká vychádzajúca hrana s kladnou kapacitou.

Počas behu algoritmu je vždy splnená prvá podmienka, lebo jediný spôsob ako spraviť vrchol v aktívnym je poslať doň tok z iného aktívneho vrcholu, ktorý si môžeme označiť u. Z u podľa indukčného predpokladu existuje cesta do zdroja a poslaním toku do nového aktívneho vrcholu v sa táto cesta neporuší, navyše v reziduálnej sieti má v tom momente hrana (v, u) kapacitu

minimálne tak veľkú ako množstvo toku poslané z u do v takže ak predĺžime $u - s$ cestu o hranu (v, u) tak získame $v - s$ cestu zo zadania vety.

Prvýkrát keď niektorý zo susedov vracia tok naspäť do vrcholu s a teda vytvorí hranu s kladnou kapacitou z vrcholu s, tak musí platiť, že výška tohto vrcholu je väčšia ako výška s, tá je ale počas behu algoritmu rovná počtu vrcholov a teda podľa vety 4.5 z tohto vrcholu neexistuje nijaká cesta do ústia, z čoho vyplíva, že cez daný vrchol nemôže tiecť tok do ústia. Počas ďalšieho behu algoritmu sa už nemôže stať, že by z tohto vrcholu vznikla v reziduálnej sieti cesta do ústia, lebo tok by doň mohol dotiecť len z vrcholov ktoré majú vyššiu výšku ako on a pre tie rovnako platí, že z nich neexistuje cesta do ústia. □

Veta 4.7. *Počas behu preflow-push algoritmu na st-sieti $G = (V, E)$, sú výšky všetkých vrcholov vždy nižšie ako $2N$.*

Dôkaz.
Pri spustení algoritmu tvrdenie platí, pretože podľa definície st-siete platí, že z každého vrchola okrem zdroja existuje cesta do t a z vety 4.4 vyplýva že tieto výšky nemôžu byť vyššie ako N, výška zdroja je N. Pre jednoduchosť ďalej predpokladajme, že výšky vo funkcií PushRelabel zvyšujeme o 1. Správnym výberom aktívnych vrcholov určených na spracovanie vieme takouto metódou odsimulovať i druhý spôsob zvyšovania výškovej funkcie v danom vrchole a teda i v tomto prípade tvrdenie platí.

Budeme uvažovať len aktívne vrcholy, keďže výška každého neaktívneho vrcholu je rovnaká, alebo o 1 vyššia, ako keď bol posledýkrát aktívny. Podľa vety 4.6 vieme, že v reziduálnej sieti existuje cesta z ľubovoľného aktívneho vrcholu do zdroja, a teda pomocou podobnej logiky ako v dôkaze vety vety 4.4 vidieť, že táto výška je nižšia ako najkratšia cesta do zdroja + výška zdroja, tj. je nutne nižšia alebo rovná ako $2N - 2$. Ako sme predtým ukázali, v prípade neaktívnych vrcholov môže byť táto výška o 1 vyššia, čo stále spĺňa kritériá vety. □

Veta 4.8. *Metóda preflow push vypočíta hodnotu maximálneho toku.*

Dôkaz.
V prvom rade potrebujeme ukázať, že algoritmus skončí činnosť v konečnom čase. Ak by bežal nekonečne dlho, tak by muselo nastať i nekonečne veľa operácí zvýšenia výškovej funkcie vo vrcholoch, lebo volanie operácie push nemôže prebehnúť viac ako konečný počet krát, bez toho aby nebola volaná

operácia relabel. Tok totiž môže tiecť len do vrcholov s nižšou výškou a v prípade že z vrcholu neodtečie, tak je volaná operácia relabel. Ak ale máme nekonečne veľa operácií relabel, tak určite existuje taký vrchol, ktorý má výšku porušujúcu vetu 4.7. Takže lubovoľná sekvencia push a relabel operácií vedie k tomu, že algoritmus zastaví a teda počet aktívnych vrcholov bude v konečnom čase 0 .

V tom prípade preflow v každom vrchole spĺňa i podmienky toku a teda po skončení vygeneruje preflow-push metóda tok. Že je maximálny vyplýva z rovnakej úvahy o vzniknutom reze ako pri dôkaze správnosti Ford-Fulkersonovho algoritmu. □

Ukážeme ešte jeden výsledok, ktorý ale nebudeme dokazovať.

Veta 4.9. *Metóda preflow push pracuje v čase* $\mathcal{O}(N^2M)$

Dôkaz.
Dôkaz je možné nájsť v [1] ako tvrdenie 7.16, alebo [4] ako tvrdenie 22.13 □

Jednotlivé algoritmy z triedy preflow-push

Podobne ako pri Ford-Fulkersonovej metóde, pri ktorej sme mali rôzne možnosti ako vyberať zlepšujúcu cestu i u preflow-push existuje viacero rôznych spôsobov ako pristupovať k vyberaniu aktívnych vrcholov, navyše môžeme rôzne zvoliť začiatočnú výškovú funkciu.

Implementovali sme prístup, kde aktívne vrcholy vyberáme z FIFO zásobníku (FIFO preflow-push algorithm) a prístup v ktorom uprednostňujeme aktívny vrchol s vyššou hodnotou výškovej funkcie (highest-label preflow-push algorithm).

Okrem týchto dvoch sa v praxi pomerne často používa capacity scaling algorithm, ktorý hľadá hrany medzi takými dvojicami vrcholov, že vrchol odkiaľ posiela tok má dostatočne vysoký rozdiel medzi prítokom a odtokom a vrchol kam hrana smeruje má tento rozdiel dostatočne malý.

4.3 Implementácia

Pri implementácií sme vychádzali z prístupu zvoleného v [4]. Vytvorený program sme napísali v C++, kde jednotlivé tokové algoritmy sú metódami triedy Graph. Implementovali sme tri rôzne algoritmy Ford-Fulkersonovej metódy

(hľadanie zlepšujúcich ciest pomocou prehľadávania do šírky, do hĺbky a hľadanie zlepšujúcich ciest s maximálnou kapacitou), dva algoritmy Preflow-push (vyberanie aktívnych vrcholov z FIFO kontajnera a vyberanie aktívnych vrcholov s najvyššou hodnotou výškovej funkcie) a experimentovali sme s rôznymi začiatočnými výškovými funkciami a oboma spomenutými spôsobmi zvyšovania vrcholu. Graf udržiavame v štruktúre `vector< list<*Edge> >`, kde inštancia triedy Edge v sebe obsahuje informácie o kapacite, aktuálnom toku, koncových vrcholoch a orientácií danej hrany. Hrana ako taká je neorientovaná, orientáciu zisťujeme pomocou metódy `Edge from(int v)`. To z toho dôvodu, že k hrane väčšinou pristupujeme bez ohľadu na jej orientáciu. Trieda Edge navyše obsahuje metódy pomocou ktorých posiela tok a zisťuje aktuálnu kapacitu na danej hrane.

Naša implementácia Ford-Fulkersonovho algoritmu hľadajúca zlepšujúce cesty s maximálnou kapacitou volá v riadku 2 v pseudokóde GenericFord-Fulkerson metódu PFS (priority first search). Táto podobne ako Dijkstrov algoritmus prechádza jednotlivé vrcholy začínajúc vo vrchole s a ukladá ich do prioritnej fronty, s tým že túto usporiadava podľa veľkosti toku, ktorý sa zo zdroja do daného vrcholu dá pretlačiť. Celkom prirodzene sa snaží najskôr spracúvať vrcholy s vyšším potenciálnym tokom. Pri nájdení nenavštíveného vrcholu tento zaradí do fronty, pričom si v nej, okrem maximálneho toku ktorý tam je schopný dostať, pamätá i prvú hranu na ceste z tohto vrcholu do zdroja. Takýmto spôsobom je potom algoritmus schopný okamžite pristupovať k nájdenej st ceste. Ľahko sa dá vidieť, že algoritmus má pomerne zložitú réžiu, ktorá je však vyvážená tým, že pomerne rýchlo nájde najširšie cesty od zdroja k ústiu, i v prípade, že väčšie množstvo ciest s minimálnou priepustnosťou je kratších. Ako však ľahko vidieť pri sieťach s jednotkovými kapacitami hrán nám tento prístup nemôže nijak pomôcť a podobá sa skôr prehľadávaniu do šírky so zbytočne veľkou réžiou.

Ostatné 2 algoritmy z tejto rodiny funguju na podobnom princípe, s tým rozdielom, že namiesto hľadania pomocou `priority_queue` volajú procedúry prehľadávania do šírky a do hĺbky.

Algoritmy z triedy Preflow-push, ktoré sme implementovali pracujú podľa pseudokódu uvedeného v danej kapitole. Najskôr zavolajú funkciu, ktorá vypočíta hodnoty výškovej funkcie, pokiaľ nie je uvedené inak, tak využívame výšky odvodené podľa vzdialenosti od ústia, ktoré nájdeme pomocou prehľadávania do šírky. Toto je jedno z miest kde výrazne využívame to, že používané hrany sú neorientované. Následne iterovaním cez všetky hrany vedúce zo zdroja pošleme tok do všetkých jeho susedov. Ďalej v závislosti od me-

tódy vytvoríme `priority_queue` alebo `queue` do ktorých vkladáme všetky aktívne vrcholy a iterujeme až do chvíle, kým existujú aktívne vrcholy. V každej iterácií vykonávame operácie push a relabel presne podľa toho ako bolo určené v pseudokóde pre generickú metódu.

Okrem metód triedy Graph sme vytvorili i niekoľko generátorov grafov, ktoré generujú jednotlivé triedy, alebo podtriedy bližšie popísané v ďalšej kapitole, kde sa k nim i krátko vyjadrujeme.

4.4 Redukcie

Definície ktoré sme v texte zaviedli môžu pôsobiť tak, že zužujú použiteľnosť uvedených algoritmov v praxi. Obmedzujeme sa na jeden zdroj a ústie v sieti, rovnako nedovolujeme kapacitné obmedzenia na vrcholoch. V tejto sekcií sa pokúsime neformálne ukázať, že model na riešenie problémov tokov v sieťach ako sme ho zadefinovali, má všeobecnejšie použitie. Táto problematika je obšírnejšie rozobraná v [4] v kapitole 22.4.

Maximálny tok na všeobecných sieťach

Siete v ktorých povolíme ľubovoľné množstvo zdrojov a ústí nazvime všeobecné siete. Potom platí nasledujúce tvrdenie.

Veta 4.10. *Problém maximálneho toku pre všeobecné siete je ekvivalentný problému pre st-siete.*

Dôkaz.
Alogritmus hľadania maximálneho toku pre všeobecné siete zjavne pracuje pre st-siete, keďže tie sú len špeciálnym prípadom všeobecných sietí. Stačí nám teda ukázať, že je všeobecný problém redukovateľný na problém na st-sieťach. Spôsob ako tento problém redukovať je nasledovný. V prípade že sieť nemá nijaký zdroj alebo ústie náš algoritmus vráti 0. V opačnom prípade si vytvoríme 2 nové vrcholy, ktoré si nazveme s a t (predpokladajme, že vrcholy s týmto menom pôvodná sieť neobsahovala), ktoré budú v st-sieti zdrojom a ústím. Ďalej zoberieme množinu všetkých zdrojov siete a pre každý vrchol v z tejto množiny pridáme hranu (s, v) ktorej priradíme kapacitu $c(s, v) = \text{outflow}(v)$. Zjavne z v riešení pôvodnej úlohy nemohol byť tok v reze $(\{s\}, V - \{s\})$ vyšší ako kapacita nami pridanej hrany, takže keď obdobne pridáme hrany smerujúce z ústí pôvodnej siete do vrcholu t,

tak maximálny tok v takto navrhnutej st-sieti je ekvivalentný maximálnemu toku vo všeobecnej sieti. □

S touto myšlienkou sme sa stretli už skôr, keď sme v úvodnej kapitole rozprávali o úlohach z oblasti bipartitného párenia. Skutočne postup ktorý sme tam popísali je špeciálnym prípadom redukcie všeobecnej siete na st-sieť. I bipartitné grafy, ktoré sme generovali sme priamo po vygenerovaní upravovali týmto spôsobom, keďže by ich úprava každým z testovaných algoritmov nezohrala nijakú rolu pri hľadaní rozdielov medzi jednotlivými algoritmami.

Podobná redukcia ako pri všeobecných sieťach sa dá previesť i pri úlohach o uskutočniteľnom toku (feasible flow), ktoré sú zadefinované nasledovne – predstavme si, že máme ku každému vrcholu priradenú váhu, ktorá sa v prípade, že je kladná interpretuje ako množstvo zásob a v prípade, že je záporná ako dopyt v danom vrchole. Úlohou je zistiť, či existuje taký tok, pre ktorý platí, že pre každý vrchol v je rozdiel $\text{outflow}(v) - inflow(v)$ rovný váhe tohto vrcholu.

Redukcia tejto úlohy je podobná, ako pri všeobecných sieťach, s tým rozdielom, že pridávané hrany nemajú kapacitu rovnú prítoku resp. odtoku, ale váhe tohto vrcholu. Je vidieť, že keď je maximálny tok v takejto sieti rovný hodnote rezu $(\{s\}, V - \{s\})$ a zároveň rezu $(V - \{t\}, \{t\})$, tak daná úloha má riešenie. Všetky vrcholy pôvodnej siete sú vnútornými vrcholmi upravenej, platí pre ne teda podmienka, že odtok vrchola je rovný prítoku doň. Tie s kladnou váhou majú prítok rovný svojej váhe, podoboné pozorovanie platí pre tie so zápornou váhou. Do (ani z) vrcholov s nulovou váhou nevedie nijaký nový vrchol, teda pre ne platí, že majú nulový prítok.

Toky s kapacitným obmedzením na vrcholoch

Veta 4.11. *Problém maximálneho toku pre siete s kapacitným obmedzením na hranách aj vrcholoch je ekvivalentný problému maximálneho toku pre st-siete.*

Dôkaz.
Podobne ako v minulom dôkaze, vyriešiť danú úlohu na st-sieti pomocou algoritmu pre všeobecnejší nie je problém, stačí, keď hodnoty kapacít na jednotlivých vrcholoch nastavíme na dostatočne vysokú hodnotu, čo pre konkrétny vrchol môže byť napríklad maximum z prítoku a odtoku daného vrcholu. Naopak, problém na sieti s kapacitným obmedzením vieme vyriešiť miernou

modifikáciou tejto siete. Pre každý vrchol v v pôvodnej sieti vytvoríme vrcholy v_1 a v_2 s tým, že hrana medzi nimi bude mať hodnotu kapacitného obmedzenia na pôvodnom vrchole a hrany vchádzajúce do v budú v upravenej sieti vchádzať do vrcholu v_1 a hrany opúšťajúce v budú opúšťať vrchol v_2. Táto úprava spôsobí, že odtok z vrcholu nebude vyšší ako jeho kapacita. Prirodzene novým zdrojom a ústím budú vrcholy s_2 a t_1, vzhľadom k pôvodným s a t. □

Toky na neorientovaných sieťach

Podobné tvrdenie ako sme dokazovali v predchádzajúcich sekciách si ukážeme i pre sieťe s neorientovanými hranami. Že sú tieto problémy ekvivalentné je dôležité pre ďalšie 2 kapitoly, kde budeme pomerne často pracovať so sieťami ktoré sú neorientované, ale reprezentujeme ich pomocou triedy pracujúcej s orientovanými grafmi.

Veta 4.12. *Problém maximálneho toku pre st-siete s neorientovanými hranami je redukovateľný na problém maximálneho toku na st-sieťach.*

Dôkaz.
K sieti s neorientovanými hranami vytvoríme st-sieť tak, že zachováme vrcholy a každej neorientovanej hrane priradíme dve orientované s opačným smerovaním a rovnakou kapacitou ako mala pôvodná hrana. Zjavne k ľubovolnému toku v pôvodnej sieti existuje tok v orientovanej, ale platí i opak. Ak máme v orientovanej sieti tok na dvoch opačne orientovaných hranách, tak na neorientovanej je s ním ekvivalentný tok v smere väčšieho z týchto dvoch tokov a v hodnote ich rozdielu. Takže maximálnemu toku v orientovanej sieti vieme nájsť nejaký tok v neorientovanej, tento je ale tiež maximálny. Ak by od neho existoval väčší tok, tak tento podľa prvej časti dôkazu vieme previesť na tok v orientovanej sieti s vyššou hodnotou ako má tok ktorý je maximálny, čo je spor. Teda ku každému maximálnemu toku v orientovanej sieti vieme jednoznačne priradiť maximálny tok v neorientovanej sieti. □

Kapitola 5

Špeciálne typy grafov

V tejto kapitole sa budeme venovať niektorým triedam grafov ktoré sa často objavuju pri riešení úloh spojených s tokmi v sieťach, zadefinujeme si ich a v ďalšej kapitole ukážeme, ako sa na týchto triedach správajú jednotlivé tokové algoritmy.

5.1 Siete malého sveta

Siete malého sveta sú modelom, ktorý sa ukázal byť úzko prepojený so sociálnymi sieťami a so štruktúrou internetu. Existuje viacero rôznych modelov, kde je dôležité aby generované grafy mali nasledujúce dve vlastnosti – nízky priemer grafu a vysokú hustotu lokálnych prepojení pri nízkej hustote prepojení vzdialených bodv.

My sme si na implementáciu vybrali model Wattsa a Strogatza (The Watts-Strogatz small-world model), kde vrcholy grafu prepájame nasledovne – vyberieme si nejaké usporiadanie vrcholov a vytvoríme neorientovanú hranu medzi každým vrcholom a k jeho susedmi v tomto usporiadaní (kde vrchol $V - 1$ susedí s vrcholom číslo 0). Takýto graf má vysoký priemer, preto k nemu s pravdepodobnosťou p pridávame náhodné hrany, čo už pri nížších pravdepodobnostiach výrazne znižuje priemer grafu.

5.2 Regulárne grafy

Definícia 5.1. Regulárny graf *(regular graph) je graf, kde každý vrchol má rovnaký stupeň. Ak je každý vrchol grafu G stupňa k, tak graf nazývame* k-

regulárny. *Špeciálne, 3-regulárny graf sa označuje* kubický.

Regulárne grafy sme generovali pomocou greedy algoritmu, kde sme pridávali hrany náhodným dvojiciam vrcholov so stupňami nižšími než k až do chvíle, kým nám neostala množina vrcholov nižšieho stupňa ako k, ktorá sa už nedala vzájomne prepojiť. Vtedy sme náhodne odstránili niekoľko hrán a pokúsili sme sa znova prepojiť tieto hrany. Zvyčajne sme po niekoľko málo odstráneniach dostali požadovaný regulárny graf.

Výhradné postavenie v teórií grafov majú kubické grafy, preto sme sa pri testoch na grafoch tejto triedy primárne zamerali na ne.

5.3 Bipartitné grafy

Definícia 5.2. Bipartitný graf *(bipartite graph)* $G = (V, E)$ *je neorientovaný graf, v ktorom vieme množinu* V *rozdeliť na dve disjunktné podmnožiny* V_1, V_2 *tak, že nijaká hrana z* E *nemá oba konce v jednej množine a* $V_1 \cup V_2 = V$.

Bipartitné grafy majú pri tokoch v sieťach dôležité postavenie, pretože úlohy z oblasti bipartitného párovania sa riešia práve na takýchto grafoch. Pre naše potreby bude postačúce pracovať s orientovaným grafom, kde si hrany zorientujeme tak, že ak $(u, v) \in E$, tak $u \in V_1$ a $v \in V_2$. Keďže v našej definícií toku nemáme povolené mať v st-sieti viacero zdrojov a ústí, tak graf ktorý používame bude mať navyše vrchol s, z ktorého budú ústiť hrany do každého vrcholu z V_1 a vrchol t do ktorého budú smerovať hrany z každého vrcholu z V_2.

5.4 Riedke a husté grafy

Husté a riedke grafy (dense and sparse graph) sú v literatúre definované len pomerne neurčito. Vo všeobecnosti sa za hustý graf považuje taký, kde je počet hrán blízky k maximalnému možnému počtu a riedky zas graf s málo hranami.

Poznámka 5.1. *Triedu graf sme implementovali tak aby bola efektívna na riedkych grafoch, keďže vo väčšine úloh o maximálnom toku sa objavuje práve tento typ grafov(v praxi napríklad cestná sieť, internetové prepojenia a podobne). Preto sú výsledky na hustých grafoch viac-menej orientačné a pri inom spôsobe uloženia grafu by mohli byť výsledné časy o čosi nižšie.*

5.5 Iné triedy grafov

Z iných tried grafov sme testovali kompletný graf (ktorý je špeciálnym prípadom hustých grafov, takže sa k nemu vzťahuje poznámka 5.1) K_n, kde každý z vrcholov má $n - 1$ susedov. Keďže pri týchto grafoch závisí počet hrán kvadraticky od množstva vrcholov, tak sme nemohli generovať príliž veľké grafy.

Poslednou triedou, ktorej sme sa v našej práci venovali boli mriežky (lattice graphs). Takýto neorientovaný graf získame tak, že si do roviny nakreslíme v pravidelných intervaloch $n \times m$ vrcholov a každý z nich prepojíme so vrcholmi s ktorými susedia na horizontálnej a vertikálnej osi. Pri týchto grafoch sa ukázalo byť veľmi dôležité, či hrany majú jednotnú kapacitu, alebo náhodnú.

Kapitola 6

Praktické testy

V tejto kapitole sa pokúsime ukázať výsledky našich pozorovaní, ku ktorým sme prišli pri testovaní jednotlivých algoritmov na rôznych grafoch. Ako sme spomenuli v úvode, nezaujíma nás len porovnávanie časov behu jednotlivých algoritmov, lebo tieto sa dosť podstatne menia už pri menších zmenách v implementácií. Budeme sa zaujímať aké maximálne toky jednotlivé algoritmy nájdu, koľko operácií počas behu algoritmu vykonajú a ako ovplyvňuje typ skúmaného grafu ich efektivitu oproti iným algoritmom.

V prvej časti tejto kapitoly sa budeme zaoberať skúmaním algoritmov iným spôsobom ako meraním času ich behu na CPU. Budeme sa snažiť zistiť koľko volaní jednotlivých procedúr algoritmy vykonajú, prípadne koľko hrán spracúvajú a namiesto množstva použitého času CPU budeme sledovať množstvo vykonaných operácií rôznych typov. Tento prístup počítania charakteristických operácií (representative operation counts) nezavedieme formálne[1], ale uvedieme si aspoň niekoľko dôvodov prečo môže byť výhodným oproti meraniu spotrebovaného času na CPU.

- Tým, že presnejšie vieme ktoré operácie sa v akom množstve vykonávajú, môžeme omnoho ľahšie hľadať bottleneck operácie v danom programe.
- Keďže skúmame len množstvo volaní operácií a nie čas závislý od množstva premenných, tak môžeme porovnávať rôzne algoritmy bežiace na rôznych počítačoch, prípadne napísané v rôznych programovacích jazykoch a skúmať zmenu proporcií volania jednotlivých metód pri zmene

[1] Viac sa o tejto téme dá dozvedieť v [1] v kaptiole 18 – Computational testing of algorithms

vstupu. Pomocou rôznych štatistických metód z takto získaných dát vieme vypočítať takzvaný virtuálny čas behu CPU, vďaka ktorému môžeme robiť experimenty na rôznych počítačoch a vzájomne ich porovnávať.

Skúmanie týmto spôsobom nám hovorí veľa o rozdieloch medzi prácou jednotlivých variácií jednej metódy (napríklad, že niektorý z algoritmov vykoná menej časovo náročnejších volaní istej procedúry), ale pre príliž veľké rozdiely medzi nami skúmanými metódami nebolo možné nájsť operácie, ktorých počet volaní by sme vzájomne porovnávali. Preto sme takto porovnávali algoritmy len lokálne v rámci jednej metódy.

Poznámka 6.1. *V nasledujúcom texte budeme v obrázkoch a tabuľkách jednotlivé algoritmy označovať skratkami – BFS, DFS a PFS budú označovať Ford-Fulkersonove algoritmy pracujúce na základe zlepšujúcich ciest, prehľadávania do hĺbky a prehľadávania do šírky. FIFO-PreflowPush je algoritmus ktorý vyberá vrcholy z fronty podľa poradia v akom prišli a HighestV-PreflowPush označuje algoritmus, kde vyberáme vrcholy s najvyššou hodnotou výškovej funkcie.*

Porovnanie algoritmov Ford-Fulkersonovej metódy

Pri porovnávaní jednotlivých implementácií algoritmov pracujúcich na báze Ford-Fulkersonovej metódy sme si všímali predovšetkým počet zlepšujúcich ciest ktoré každý z algoritmov počas svojho behu našiel, ich priemernú dĺžku a počet hrán ktoré počas behu skúmal.

	počet zlepšujúcich ciest	priemerná dĺžka cesty	počet hrán
bfs	12	3.83333	5075
dfs	49	13.6327	15776
pfs	6	7.33333	11395

Tabuľka 6.1: Tabuľka počtu operácií jednotlivých algoritmov z triedy Ford-Fulkerson na náhodnom grafe.

V tabuľke 6.1 vidíme výsledky Ford-Fulkersonovej metódy na náhodnom grafe s 60 vrcholmi a pravdepodobnosťou prepojenia dvoch vrcholov 0.1 a náhodnou hodnotou kapacity jednotlivých rán v rozmedzí 1 až 600. Výsledný

tok mal hodnotu 1483. Ako vidíme z tabuľky, algoritmus hľadajúci cesty s najväčšou kapacitou, potrebuje na nájdenie toku najmenej augmentácií, prehľadávanie do šírky nachádza väčšie množstvo kratších ciest a prehľadávanie do hĺbky na tomto type grafov spracúva výrazne väčšie množstvo vrcholov než predchádzajúce dve metódy.

Porovnanie algoritmov metódy preflow-push

Medzi algoritmami z tejto triedy sme celkom prirodzene pozorovali počet poslaní toku do nižšieho z vrcholov a zvýšení výškovej funkcie vo všetkých bodoch. Podobne ako pri Ford-Fulkersonovej metóde sme zisťovali i celkový počet hrán ktoré daný algoritmus prezrel počas hľadania maximálneho toku.

	volania push	volania relabel	počet hrán
FIFO preflow	13083	2995	75363
Highest Vertex preflow	15008	2938	81192

Tabuľka 6.2: Tabuľka počtu operácií jednotlivých algoritmov z triedy Preflow-Push na náhodnom grafe.

V tabuľke 6.1 vidíme výsledky preflow-push metódy na rovnakom grafe ako bol použitý v tabuľke 6.1. Pri metóde Preflow-Push vidíme výrazne menší rozdiel medzi počtom volaní jednotlivých metód v oboch použitých algoritmoch. Zároveň však na podobných grafoch preukazuje výrazne vyššie rozdiely v počte jednotlivých operácií veľmi závislé od toho, koľko toku sa mu podarí dostať do ústia. V prípade, že nepotrebuje vracať nijaký tok do zdroja (tj. v prípade, že je výsledný tok rovný rezu $(\{s\}, V - \{s\})$), tak vykonáva výrazne menej operácií (pri grafoch s podobným počtom hrán a vrcholov niekedy len okolo 1000 prezrených hrán), než v prípadne, že nejaký tok do zdroja vracia.

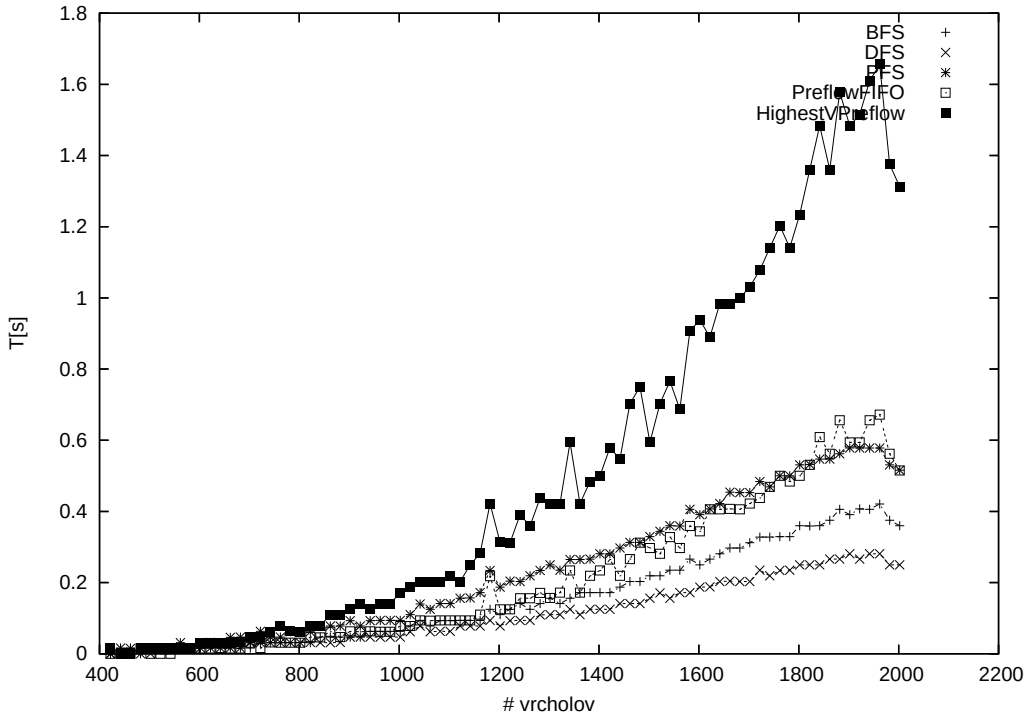
Ďalej sme skúmali i variáciu preflow-push algoritmu, ktorý namiesto najvyšších vrcholov primárne vyberá zpomedzi vrcholov vždy najnižší, ale výsledné časy behu tohto algoritmu sa neukazovali byť výrazne odlišné od metódy vyberajúcej najvyšší vrchol a vo väčšine testov na väčších grafoch sa ukazoval byť o čosi málo pomalší.

Čo sa týka zmeny počiatočnej výškovej funkcie, skúmali sme zmenu za funkciu ktorá je všade okrem zdroja nulová a výsledné hodnoty sa veľmi nelíšili od algoritmov ktoré využívali sofistikovanejšiu výškovú funkciu.

Vo všeobecnosti sa ukázalo, že algoritmy založené na tejto metóde majú tendenciu sa vzájomne menej líšiť v množstve spotrebovaného času na vstupných grafoch z rôznych tried a rôznej veľkosti.

Porovnanie časov jednotlivých algoritmov na rôznych grafoch

V tejto sekcií si ukážeme aké rozdiely boli medzi časmi jednotlivých algoritmov na rôznych vstupných grafoch. Jednotlivé dáta, ktoré je možné vidieť vizualizované na obrázkoch v tejto sekcií reprezentujú priemerný čas behu našich implementácií jednotlivých algoritmov na grafoch rôznej veľkosti. Každý bod v tomto grafe reprezentuje medián z meraní časov na jedenástich rôznych grafoch danej veľkosti. Na x-ovej osi je počet hrán vstupu a na y-ovej čas behu mediánu v sekundách. Všetky algoritmy bol testované naraz na rovnakých vstupných dátach.

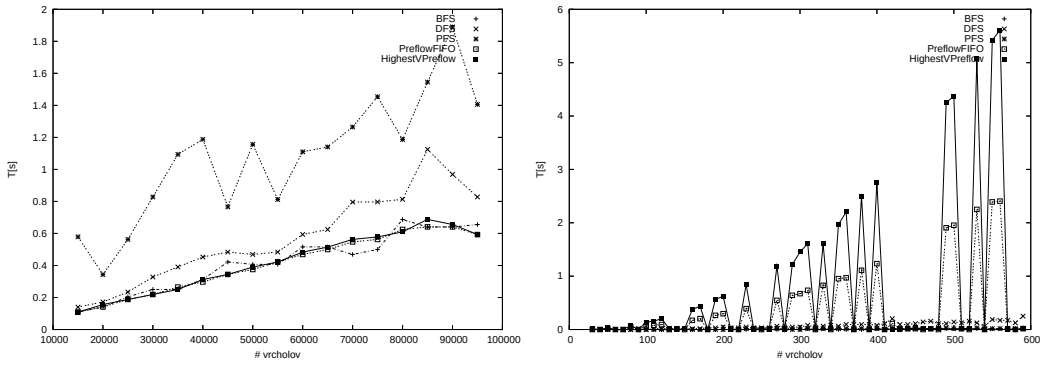


Obr. 6.1: Porovnanie algoritmov na bipartitných grafoch

Na obrázku 6.1 vidieť časy jednotlivých algoritmov na bipartitných grafoch rôznej veľkosti, s dvoma rovnako veľkými bipartíciami o veľkosti $N/2$, pravdepodobnosťou prepojenia hrany 0.0005 a jednotkovou kapacitou hrán. Hodnoty jednotlivých tokov sa pohybovali v závislosti od veľkosti vstupu od 10 po 400.

Ako vidieť, algoritmy HighestVertex Preflow-push a Ford-Fulkerson PFS pracujú v tomto meraní omnoho pomalšie než ostatné algoritmy z ich tried. Dôvodom je v prvom prípade predovšetkým nízka dĺžka jednotlivých ciest, kvoli ktorej sú výškové funkcie jednotlivých vrcholov veľmi podobné, v druhom prípade je problémom predovšetkým jednotková kapacita hrán a to, že všetky cesty vedúce od zdroja k ústiu majú rovnakú dĺžku. Preto ani jeden z týchto algoritmov nezískava nijakú výhodu oproti iným, naopak práca s prioritnou frontou im výrazne zvyšuje čas nutný na vykonanie základných operácií. Nízka pravdepodobnosť prepojenia hrán spôsobuje, že je do $N/2$ vrcholov poslaný tok, z ktorého sa v asi tretina musí vrátiť naspäť do zdroja, čo môže byť jeden z dôvodou prečo majú algoritmy z triedy preflow-push na týchto i niektorých ďalších grafoch výrazne horšie celkové časy.

Spomenuli sme, že jednou z premenných pri efektivite algoritmov je práve to, či majú hrany jednotkovú kapacitu, preto budeme ďalej uvádzať pri niektorých triedach grafov výsledky pri jednotkových (resp. jednotných) i náhodných kapacitách.

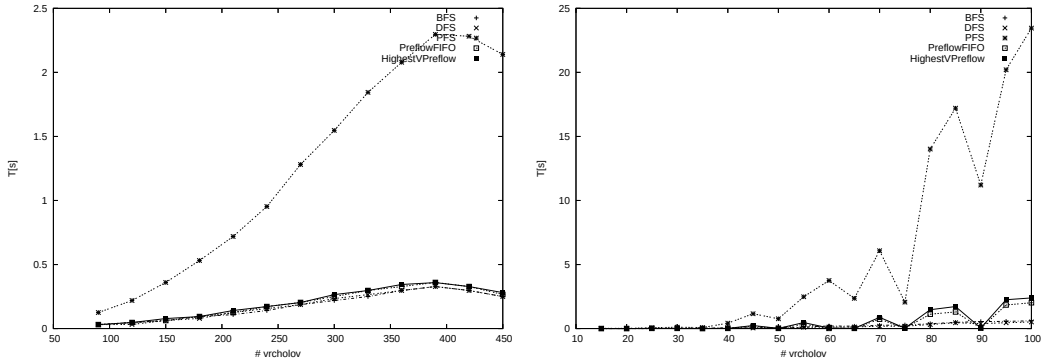


(a) jednotková kapacita

(b) náhodná kapacita

Obr. 6.2: Porovnanie algoritmov na kubických grafoch

Časy jednotlivých algoritmov na kubických grafoch možno vidieť na obrázku 6.2 , kde v časti (a) vidíme časy na grafoch s jednotkovou kapacitou. Pri týchto grafoch mal tok vo všetkých prípadoch hodnotu 3. Ako vidno preflow-push algoritmy a prehľadávanie do hĺbky v tomto prípade dosahovali lepšie časy než zvyšné dva algoritmy, čo celkom prirodzene vyplýva už z našich predchádzajúcich pozorovaní, že tieto algoritmy si dobre počínajú v prípade keď nemusia vracať veľké množstvo toku z ústia do zdroja resp. u PFS v prípade, že cesty ktoré sú najkratšie sú zároveň najvýhodnejšími, čo je pomerne častá situácia pri grafoch s jednotkovou kapacitou. Obrázok (b) zas ukazuje ako veľmi sa môžu rôzniť časy behu algoritmov preflow-push metódy pre rôzne vstupy. Priebeh časov ktorý vidíme u kubických grafov je veľmi podobný priebehu ktorý vidieť u mriežkových grafoch.



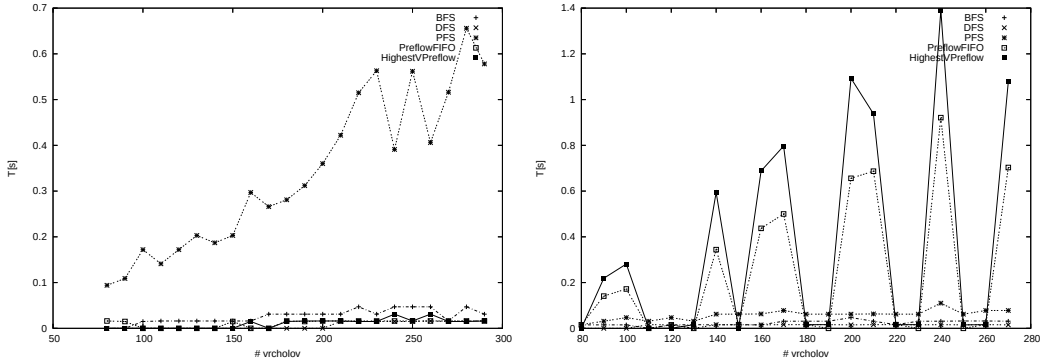
(a) jednotná kapacita 100

(b) náhodná kapacita 1-200

Obr. 6.3: Porovnanie algoritmov na kompletných grafoch

Pri menších kompletných grafoch, pri jednotnej, ako i náhodnej, kapacite až tak výrazne nezáleží na výbere algoritmu, s výnimkou hľadania zlepšujúcich ciest pomocou PFS, u ktorého opäť práca s prioritnou frontou znamenala zbytočné predražovanie operácie augument.

Na obrázku 6.4 je možné vidieť časy jednotlivých algoritmov na sieťach malého sveta. V prvom prípade sme zvolili sieťe s jednotnou kapacitou 10, s prepojením deviatich susedov na oboch stranách a pravdepodobnosťou prepojenia 0.005. V druhom prípade sme zvolili, náhodnú kapacitu v rozmedzí 1 až 10, 5 prepojených susedov a rovnakú pravdepodobnosť prepojenia.

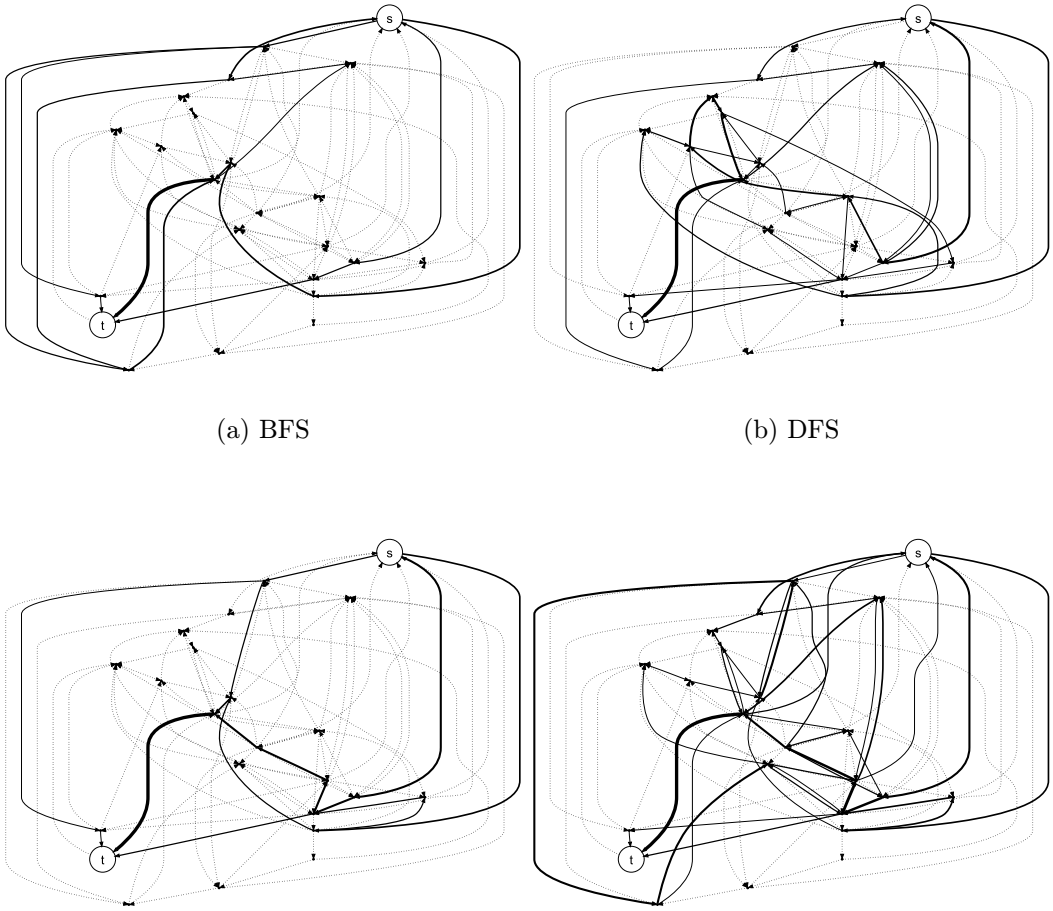


(a) jednotná kapacita 10

(b) náhodná kapacita 1-10

Obr. 6.4: Porovnanie algoritmov na sieťach malého sveta

Tokové algoritmy niekedy generujú výrazne sa odlišujúce maximálne toky, Takéto porovnanie na náhodnom grafe môžeme vidieť na obr. 6.5 . Často pre nás môže byť výhodné nachádzať tok s čo najmenším počtom hrán, preto i toto môže byť jedným z atribútov na ktoré sa treba zamerať pri výbere vhodného algoritmu pre danú úlohu. Vo všeobecnosti sa pri jednotných alebo jednotkových kapacitách ukazuje ako vhodný kandidát Ford-Fulkersonov algoritmus prehľadávajúci sieť do šírky a pri sieťach s náhodnými kapacitami Ford-Fulkersonov algoritmus prehľadávajúci pomocou metódy PFS.



(a) BFS

(b) DFS

(c) PFS

(d) Preflow-Push

Obr. 6.5: Porovanie maximálnych tokov na náhodnom grafe

Kapitola 7

Záver

V tejto práci sme sa zamerali na jednotlivé algoritmy používané pri riešení úlohy o maximálnom toku. V teoretickej časti práce sme ukázali niekoľko úloh ktoré sa dajú riešiť pomocou tokov v sieťach. Zaviedli sme jednotlivé algoritmy a ich triedy, ukázali sme, že algoritmy pracujú korektne a nachádzajú maximálny tok. V praktickej časti sme tieto algoritmy implementovali a testovali na nami generovaných grafoch.

Z nameraných hodnôt sa zdá, že v prípade jednotkovej, resp. jednotnej kapacity hrán je výhodné voliť algoritmy, ktoré pracujú s dátovými štruktúrami nenáročnými na čas, zatiaľ čo implementácia algoritmov pomocou prioritnej fronty z STL sa ukázala vo veľkej časti prípadov pomerne časovo náročnou. Pri grafoch s náhodnou kapacitou sa vo väčšine prípadov naša implementácia preflow-push metód ukázala ako pomerne ťažkopádna, navyše má omnoho ťažšie predvídateľný výsledný čas na grafe istej veľkosti a triedy. Naopak výhodné je tieto algoritmy použiť pri hľadaní maximálnych tokov v grafoch kde je hodnota rezu rovná rezu medzi zdrojom a ostatnými vrcholmi – v tomto prípade sa ukazuje výhodnou i Ford-Fulkersonova metóda prehľadávajúca do hĺbky.

Zo smerov, ktorými by sa v budúcnosti dalo naviazať na túto prácu, celkom prirodzene pripadajú do úvahy snahy o efektívnejšie implementácie predovšetkým v triede preflow-push algoritmov. V našej práci sme sa snažili o implementáciu algoritmov, ktoré nestrácajú na všeobecnosti a teda fungujú na všetkých sieťach spĺňajúcich nami zadefinované podmienky. Jednou z možností ako nájsť algoritmus, ktorý dobre pracuje na konkrétnom type grafu efektívne, je znížiť jeho všeobecnosť a počítať s podmienkami ktoré v danom type grafu platia. Ďalšou možnosťou je skúmať úlohy o toku s mi-

nimálnou cenou, ktoré sú prirodzeným rozšírením problému maximálneho toku, prípadne iné tokové úlohy v ktorých sa úloha o maximálnom toku rieši ako podproblém.

Zoznam obrázkov

6.1 Porovnanie algoritmov na bipartitných grafoch 30
6.2 Porovnanie algoritmov na kubických grafoch 31
6.3 Porovnanie algoritmov na kompletných grafoch 32
6.4 Porovnanie algoritmov na sieťach malého sveta 33
6.5 Porovanie maximálnych tokov na náhodnom grafe 34

Zoznam tabuliek

6.1 Tabuľka počtu operácií Ford-Fulk. 28
6.2 Tabuľka počtu operácií PreflowPush 29

Literatúra

[1] Ahuja, R., Magnanti, T., Orlin, J. *Network Flows: Theory, Algorithms, and Applications* Prentice Hall, 1993. 864 s. ISBN 013617549X

[2] Cormen, T., Leiserson, Ch., Rivest, R., Stein, C. *Introduction to Algorithms, 2nd Edition* MIT Press, 2001. 1180 s. ISBN 0-262-53196-8

[3] Diestel, R. *Graph Theory, Electronic Edition 2005* Springer-Verlag Heidelberg, New York, 2005. 410 s.

[4] Sedgewick, R. *Algorithms in C++ Part 5: Graph Algorithms, 3rd Edition* Addison Wesley Professional, 2003. 528 s. ISBN 0-201-36118-3