Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gregor Ajdič

Algoritmi problema pretokov

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Borut Robič

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Predstavite najsodobnejše algoritme za reševanje problema pretokov v omrežjih. Razložite glavne zamisli in pojme, na katerih temeljijo ti algoritmi. Njihovo delovanje ilustrirajte na primerih. Primerjajte njihove časovne zahtevnosti in razmislite o njihovi praktični uporabnosti.

Zahvaljujem se mentorju prof. dr. Borutu Robiču za vso pomoč pri izdelavi diplomske naloge. Zahvalil bi se tudi svoji družini za vso podporo, ki so mi jo nudili med študijem.

Posvečam svoji babici.

Kazalo

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
V	vertex	vozlišče
E	edge	povezava
G	graph	graf
R	residual graph	rezidualni graf
G_f	residual network	rezidualno omrežje
G^{ab}	abundance graph	obilni graf

Povzetek

Naslov: Algoritmi problema pretokov

Avtor: Gregor Ajdič

Predpostavimo, da je naš namen prenesti nek material od izvora do ponora preko vozlišč. Na naši poti imamo povezave, ki imajo maksimalno kapaciteto. Naš cilj je ugotoviti, največ koliko materiala lahko prenesemo od izvora do ponora, brez da bi kršili kapacitetne omejitve. Reševanje tega problema je bilo sprva možno s splošnimi tehnikami linearnega programiranja, kasneje pa so se začeli razvijati številni drugi algoritmi. Nekateri izmed njih uporabljajo pristope in metode, ki so zanimivi s teoretičnega vidika, drugi pa so primernejši za uporabo v praksi. Cilji diplomskega dela so predstavitev glavnih idej teh algoritmov, njihova analiza in prikaz nekaterih praktičnih primerov. Želimo ugotoviti, kateri algoritem je trenutno najhitrejši.

Ključne besede: pretok, omrežje pretokov, rezidualno omrežje, nenasičene poti, rezi, blokiranje poti, dinamična drevesa, metoda potiska in ponovnega označevanja, obilni graf.

Abstract

Title: Network flow algorithms

Author: Gregor Ajdič

Let us assume that it is our intention to transfer material from source to sink through the nodes of a graph. Each edge has a certain capacity. Our goal is to determine how much material can be transferred from the source to the sink without violating the capacity limit. Initially basic linear programming techniques were used to solve this problem, but later many other algorithms were discovered. Some of them use approaches and methods that are interesting from a theoretical point of view, while others are more suitable for practical use. The aims of the thesis are the presentation of these algorithms, their analysis and the display of basic examples. We want to figure out which algorithm is currently the fastest in solving this problem.

Keywords: flow, network flow, residual network, augmenting paths, cuts, blocking path, dynamic trees, push relabel method, abundance graph.

Poglavje 1

Uvod

Predstavljajmo si, da gledamo zemljevid kot usmerjen graf, ki nam pomaga najti najkrajšo pot med začetno in končno točko. Podobno si lahko predstavljamo tudi usmerjen graf, ki ga interpretiramo kot omrežje pretokov. Z njim lahko rešujemo različne probleme prenosa oziroma pretoka nekega materiala od izvora do ponora. Material se konstantno generira na izvoru in z enako hitrostjo porablja v ponoru. Primeri takšnega pretoka so na primer tekočina, ki se pretaka skozi cevi, električni tok, ki potuje skozi električna omrežja, in nenazadnje informacije, ki potujejo skozi komunikacijska omrežja. Vsako usmerjeno povezavo v takšnem omrežju si lahko predstavljamo kot cev ali žico, skozi katero se lahko pretoči omejena količina materiala v nekem času (na primer 200 litrov tekočine na uro). Vozlišča v grafu predstavljajo spoje, ki pa za razliko od izvora in ponora ne shranjujejo materiala, pač pa ta preko njih le potuje. Povedano z drugimi besedami, količina materiala, ki vstopi v vozlišče, mora biti enaka količini materiala, ki izstopi iz vozlišča. S problemom maksimalnega pretoka želimo torej ugotoviti maksimalno količino materiala, ki ga lahko pošljemo od izvora do ponora, brez da bi kršili kapacitetne omejitve samega omrežja. Pri problemu minimalnega reza pa želimo določiti množico cevi z najmanjšo skupno kapaciteto, katerih odstranitev bi povzročila, da bi bila izvor in ponor nepovezana.

To diplomsko delo je namenjeno pregledu in analizi algoritmov, ki rešujejo

problem pretokov. Pri tem se bomo omejili le na osnovne algoritme in ne bomo obravnavali posebnih primerov. Motiv za diplomsko delo je bila želja raziskati algoritme, ki rešujejo problem maksimalnega pretoka, ter predstaviti njihove glavne ideje in uporabljene metode. Skozi zgodovino so avtorji predstavili številne algoritme. Nekateri izmed njih so idealni za predstavitev obravnavanega področja, ker uporabljajo enostavne zamisli, drugi uporabljajo številne metode in pristope, ki jih naredijo zanimive predvsem s teoretičnega vidika, spet tretji pa so uporabni v praksi, kar je tudi glavni cilj algoritmov.

Cilj diplomske naloge pa je predstavitev algoritmov, ki s svojimi idejami vpeljejo neko pomembno novost na obravnavanem področju. Ker obstaja več algoritmov, ki rešujejo naš problem, v tej diplomski nalogi ne bomo obravnavali vseh, pač pa bomo predstavili le najpomembnejše. Želimo odgovoriti na vprašanje, kateri algoritem je trenutno najhitrejši.

Z opisi, analizami časovne zahtevnosti ter nekaterimi praktičnimi primeri je diplomsko delo namenjeno predvsem bralcem, ki jih področje pretokov zanima, oziroma tistim, ki bi se v prihodnje lotili raziskovanja na tem področju in bi jim diploma služila kot pomoč pri njihovem delu.

Poglavje 2

Predstavitev in definicije

Kot je opisano v [1], bomo najprej predstavili omrežje pretokov z definicijo iz teorije grafov, predstavili bomo pretok in definirali problem maksimalnega pretoka.

2.1 Omrežje pretokov

Omrežje pretokov $G = (V, E)$ je usmerjen graf vozlišč V in povezav E, v katerem ima vsaka povezava $(u, v) \in E$ nenegativno kapaciteto, ki jo označimo kot $c(u, v) \geq 0$. Zahteva se tudi, da če v E obstaja povezava (u, v), potem ne sme obstajati povezava (v, u). Prav tako ne dovolimo zank, tj. povezav, ki imajo začetek in konec v istem vozlišču. Omrežje vsebuje dve vozlišči, ki ju imenujemo izvor s ter ponor t. Predpostavimo, da leži vsako vozlišče znotraj omrežja na neki poti med izvorom in ponorom. Ker ima vsako vozlišče z izjemo izvora vsaj eno vhodno povezavo, velja $|E| \geq |V| - 1$.

2.2 Pretok

Če je torej $G = (V, E)$ omrežje pretokov s kapaciteto c, izvorom s ter ponorom t, je *pretok* v G v tem primeru funkcija $f : V \times V \to \mathbb{R}$, ki izpolnjuje sledeči zahtevi:

Omejitev pretoka: Za vse $u, v \in V$ zahtevamo $0 \leq f(u, v) \leq c(u, v)$.

Tok od enega vozlišča do drugega mora biti nenegativen in ne sme preseči dane kapacitete.

Ohranjanje pretoka: Za vse $u \in V - \{s, t\}$ zahtevamo

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

Skupni tok, ki prihaja v vozlišče (z izjemo izvora in ponora), mora biti enak toku, ki odhaja iz vozlišča.

Če $(u, v) \notin E$, potem ne more biti nobenega pretoka od u do v, zato je $f(u, v) = 0$.

Poglavje 3

Ford-Fulkerson

V tem poglavju bomo predstavili Ford-Fulkersonovo metodo za reševanje problema maksimalnega pretoka. Bralec se tu najverjetneje vpraša, zakaj jo imenujemo metoda in ne algoritem. Odgovor na to je, ker omogoča različne implementacije in s tem posledično različen čas izvajanja. Sam problem iskanja maksimalnega pretoka je poseben primer linearnega programiranja [3], in ga je mogoče rešiti s pomočjo splošnih tehnik linearnega programiranja. Ena izmed tehnik je *simplex metoda* , predstavljena v članku [4].

V članku [2] se Ford in Fulkerson zanašata na tri ideje, ki so pomembne tudi za številne druge algoritme pretoka: rezidualna omrežja, nenasičene poti in reze. Te ideje so ključne za izrek maksimalnega pretoka in minimalnega reza, ki sta ga s pomočjo metode tudi dokazala.

3.1 Rezidualno omrežje

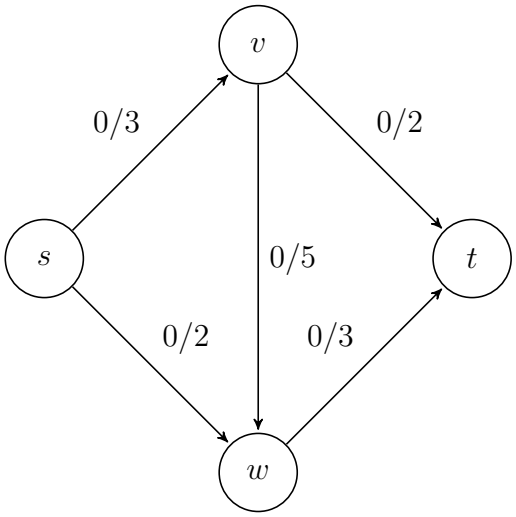
Izbira nenasičenih poti je zelo pomembna pri iskanju optimalne rešitve, z napačno izbiro poti lahko namreč prehitro blokiramo pretok. Takšni situaciji se lahko izognemo z uporabo *rezidualnega omrežja* , ki nam omogoča, da prekličemo operacijo (nam dovoli, da pošljemo tok v obratni smeri povezave, kar izniči operacijo prejšnje iteracije). *Rezidualni graf* R omrežja G ima isto

množico vozlišč kot G in za vsako povezavo v grafu G doda:

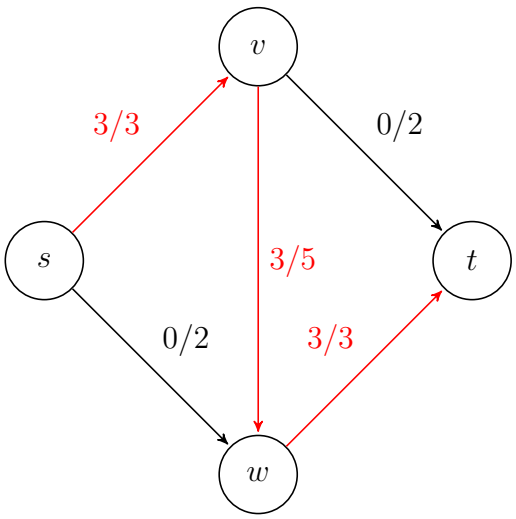
- pozitivno povezavo $e' = (u, v)$ s kapaciteto $c_e - f_e$, če velja $c_e - f_e > 0$;
- negativno povezavo $e'' = (v, u)$ s kapaciteto f_e, če velja $f_e > 0$.

.

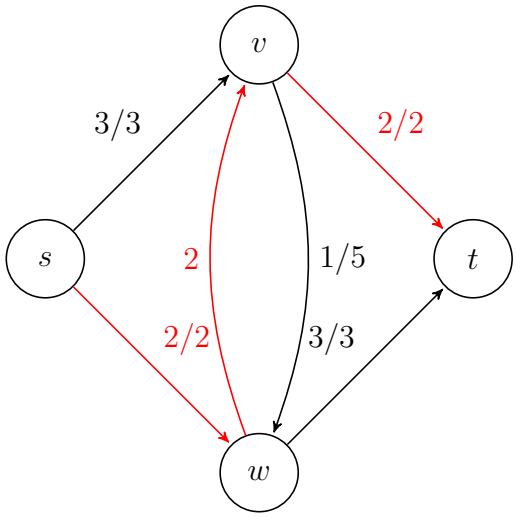
Poglejmo si na primeru. Imamo podan sledeči graf:



Če v iteraciji izberemo pot $s \to v \to w \to t$, bomo blokirali pretok, saj ne bo možno več poiskati nenasičene poti med izvorom in ponorom in posledično ne bomo dobili optimalne rešitve.

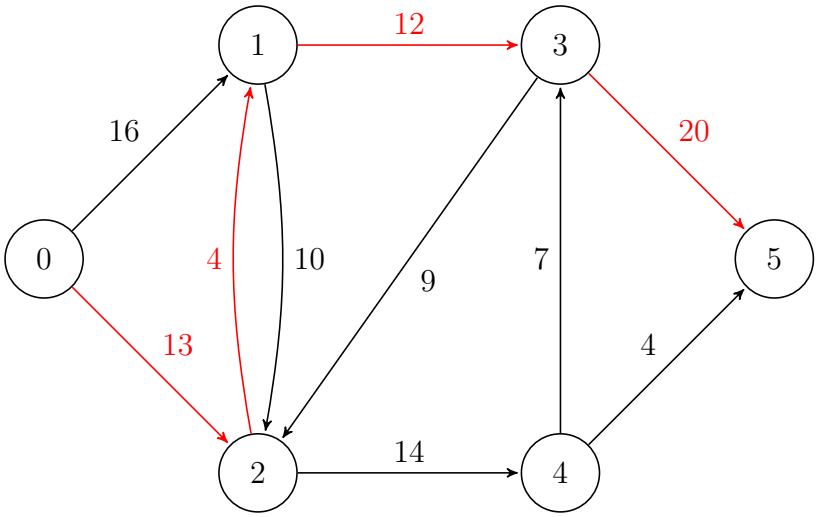


Če pa uporabimo rezidualen graf, lahko povezavo (v, w) predstavimo kot pozitivno povezavo, ki pošlje 3 enote toka od vozlišča v do w, in negativno povezavo, ki pošlje 2 enoti toka od vozlišča w do v. V tem primeru lahko pošljemo 2 enoti toka od izvora do vozlišča w, nato pošljemo po negativni povezavi 2 enoti toka od vozlišča w do v in 2 enoti toka od vozlišča v do t.



3.2 Nenasičene poti

Če imamo omrežje $G = (V, E)$ in tok f, potem je *nenasičena pot* p preprosto pot od izvora s do ponora t v rezidualnem omrežju G_f. Pretok na povezavi (u, v) nenasičene poti lahko povečamo za največ $c_f(u, v)$, brez da bi kršili omejitev pretoka.



Poglejmo si na primeru. Pot na zgornji sliki, ki je označena z rdečo barvo, je ena izmed nenasičenih poti. Pretok na vsaki povezavi te poti lahko povečamo za največ 4 enote, brez da bi kršili omejitev pretoka, saj je najmanjša kapaciteta, ki bo nasitila povezavo, $c_f(2,1) = 4$.

3.3 Rezi

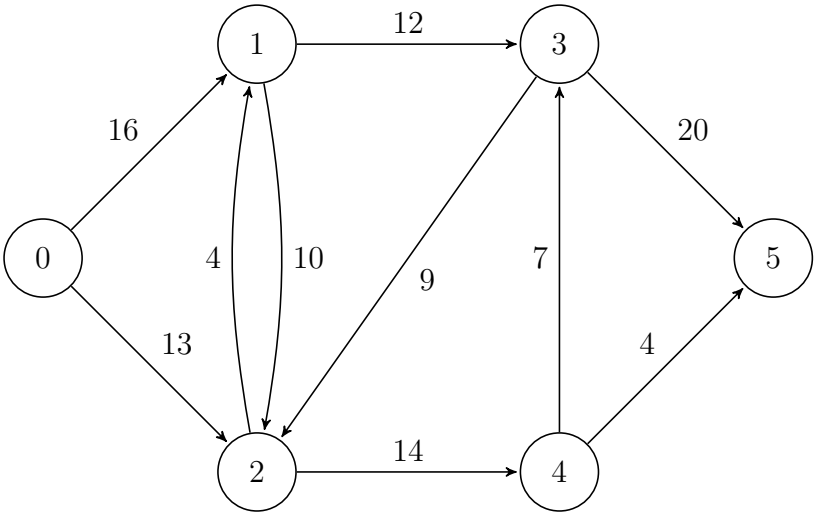
Ford-Fulkersonova metoda povečuje tok na nenasičenih poteh, dokler ne najde maksimalnega pretoka. A kako vemo, da smo dejansko našli maksimalen pretok, ko se algoritem zaključi? Ford in Fulkerson sta želela dokazati *izrek o maksimalnem pretoku in minimalnem rezu* , ki pravi, da je pretok maksimalen, če ne obstaja več nenasičena pot. *Rez* (S,T) grafa $G = (V, E)$ je particija V v S in $T = V - S$, kjer mora veljati, da izvor pripada S, ponor pa T. *Minimalni rez* v omrežju je rez, katerega kapaciteta je minimalna med vsemi rezi v omrežju. *Kapaciteto* definiramo kot:

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$$

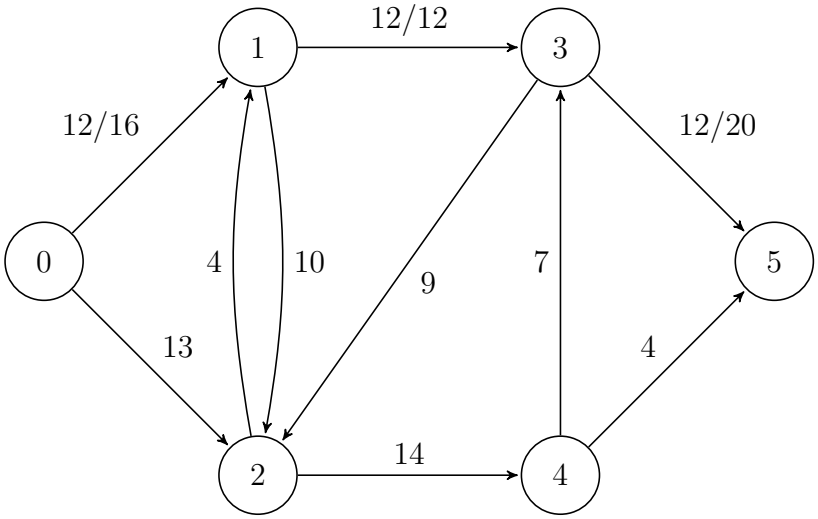
.

3.4 Primer izvajanja

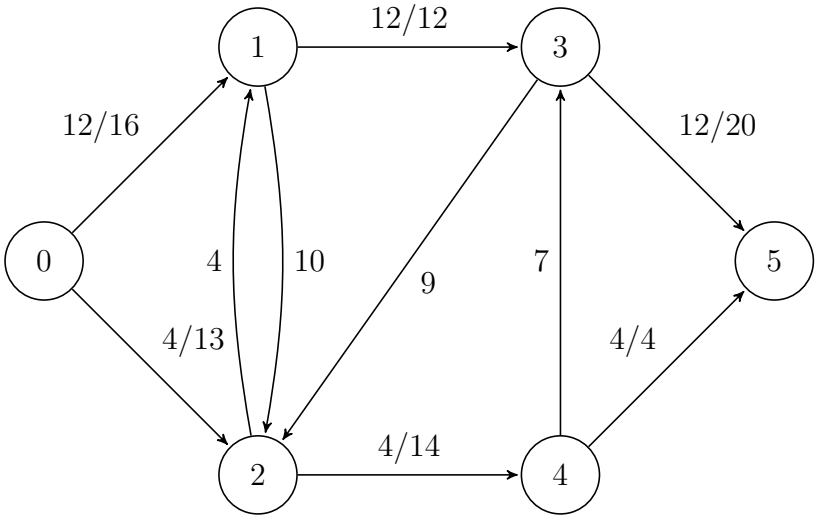
V tem podpoglavju bomo predstavili eno izmed možnih izvajanj metode na konkretnem primeru. Podan imamo graf G z izvorom v vozlišču 0 in ponorom v vozlišču 5. Ugotoviti želimo kolikšen je maksimalen tok, ki ga lahko prenesemo od izvora do ponora. Na vsaki povezavi imamo zapisano kapaciteto toka, ki ga lahko pošljemo skozi, preden le-ta postane nasičena. Najprej moramo najti nenasičeno pot, med vozliščema 0 in 5.



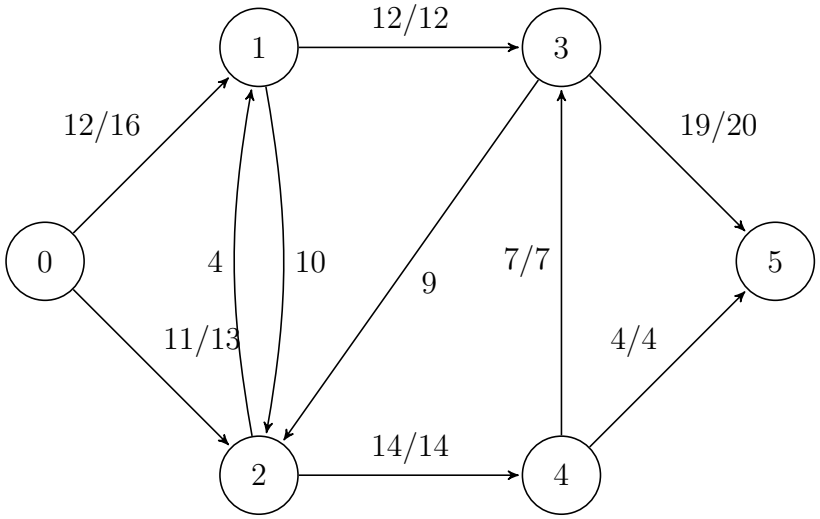
V tem primeru bomo vzeli pot $0 \to 1 \to 3 \to 5$. Če začnemo v vozlišču 0, vidimo, da lahko skozi povezavo do vozlišča 1 prenesemo maksimalno 16 enot toka, ker pa povezava med 1 in 3 ne more prenesti toliko toka, bo maksimum 12. Med povezavo 3 in 5 lahko prenesemo maksimalno 20 enot toka, ker pa iz prejšnjega vozlišča prejmemo 12 enot, graf sedaj posodobimo.



Ta pot je sedaj nasičena, saj med vozliščema 1 in 3 ne moremo prenesti več toka, zato najdemo drugo nenasičeno pot $0 \to 2 \to 4 \to 5$. Tu vidimo, da med vozliščema 4 in 5 lahko prenesemo maksimalno 4 enote toka. S to vrednostjo posodobimo svoj graf.



Tudi ta pot je sedaj nasičena, obstaja pa še ena, ki lahko prepelje tok med vozliščema 0 ter 5. Ta pot je $0 \to 2 \to 4 \to 3 \to 5$. Vidimo, da je med vozliščema 4 in 3 maksimalna kapaciteta toka, ki ga lahko prenesemo, 7, s čimer posodobimo svoj graf.



Sedaj ne obstaja več nobena nenasičena pot, preko katere bi lahko prenesli tok med izvorom 0 ter ponorom 5. Maksimalen pretok, ki ga lahko pošljemo od izvora do ponora, je torej vsota kapacitet vseh povezav, ki gredo v ponor. V tem primeru dobimo 19 enot toka od vozlišča 3 ter 4 enote od vozlišča 4, kar nam da maksimalen pretok 23.

Bralec se tukaj verjetno vpraša, na kakšen način izbiramo poti med 0 in 5. Na voljo imamo namreč več drugih, kot na primer:
$0 \to 1 \to 3 \to 2 \to 4 \to 5$
$0 \to 2 \to 1 \to 3 \to 5$
$0 \to 2 \to 4 \to 3 \to 5$
$0 \to 1 \to 3 \to 5$

Kot je bilo omenjeno na začetku tega poglavja, Ford-Fulkersonova metoda namreč ne definira, na kakšen način naj se najde nenasičena pot. Definira le, da se algoritem izvaja, dokler ni več nobene nenasičene poti med izvorom in ponorom. To nas pripelje do zaključka, da Ford-Fulkersonovega postopka ne moremo imenovati algoritem, pač pa metoda, saj ne definira, na kakšen način naj se iščejo nenasičene poti, hitrost algoritma pa lahko variira glede na izbiro poti, ki smo jo določili v implementaciji. To je nekaj, kar sta opazila tudi Edmonds in Karp ter dodala v izboljšani algoritem.

3.5 Časovna zahtevnost

Cormen v [1] razloži, da je časovna zahtevnost izvajanja algoritma $O(Ef^*)$, pri čemer f^* predstavlja maksimalen pretok v transformiranem omrežju, število korakov iskanja nenasičene poti pa je največ f^*, saj se vrednost toka poveča za vsaj eno enoto v vsaki iteraciji.

Poglavje 4

Edmonds-Karp

Za razliko od Ford-Fulkersonove metode Edmonds-Karpov postopek [12] dejansko lahko imenujemo algoritem. Gre za implementacijo Ford-Fulkersonove metode, kjer sta Edmonds in Karp definirala, na kakšen način naj se iščejo nenasičene poti. Torej $G = (V, E)$ predstavlja graf oziroma mrežo in $n = |V(G)|$ ter $m = |E(G)|$. Če Ford-Fulkersonov algoritem za iskanje maksimalnega pretoka izvajamo tako, da v vsaki iteraciji iščemo najkrajšo pot med izvorom s in ponorom t, potem je število iteracij maksimalno nm. Glede na to, da iskanje najkrajše poti lahko izvedemo v času $O(n+m)$, tako da uporabimo *iskanje v širino* [7], in ker je $m = \Omega(n)$ (saj predvidevamo, da je vsako vozlišče na neki poti med s in t, sicer bi ga lahko izbrisali iz grafa), ugotovimo, da je skupni čas izvajanja enak $O(nm^2)$. Ključno pri opazovanju tega algoritma je, da se dolžina najkrajše poti med izvorom in ponorom med izvajanjem ne zmanjšuje, pač pa se slejkoprej poveča.

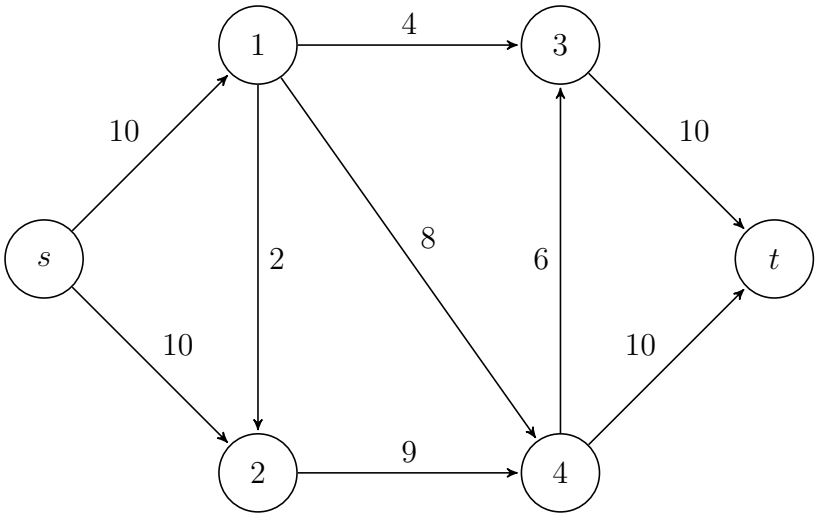
Poglavje 5

Dinitz

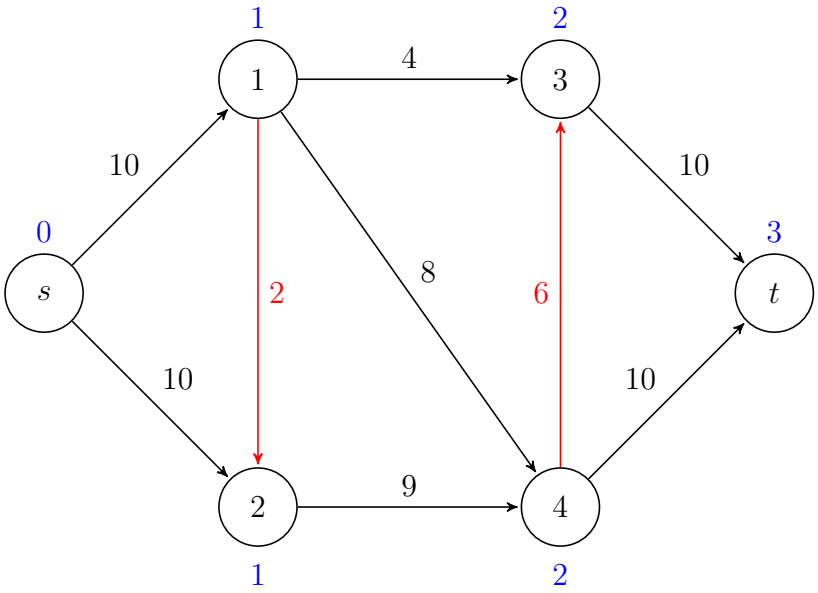
Pri Edmonds-Karpovem algoritmu se uporabi iskanje v širino [7], da najdemo nenasičene poti. Pri Dinitzevem algoritmu pa se iskanje v širino uporablja za preverjanje, če je možno poslati več toka in za konstruiranje *nivojskega grafa*. V nivojskem grafu je vsakemu vozlišču dodeljeno število, ki predstavlja nivo. Nivo je v tem primeru najkrajša razdalja (število povezav) vozlišča od izvora. Ko je nivojski graf skonstruiran, skozenj pošljemo več tokov. To je tudi razlog, zakaj algoritem deluje hitreje kot Edmonds-Karpov algoritem. Pri Edmonds-Karpovem algoritmu lahko namreč pošljemo le en tok v iteraciji, ki je najden z iskanjem v širino. Pretok je *blokiran*, če noben tok ne more biti poslan preko nivojskega grafa oziroma če ne obstaja več pot med izvorom s in ponorom t, ki bi nivojska vozlišča povezovala v naraščajočem vrstnem redu 0, 1, 2 ...

5.1 Primer izvajanja

V tem podpoglavju bomo na primeru predstavili postopek iskanja maksimalnega pretoka z uporabo Dinitzevega algoritma. Naš začetni graf izgleda tako:



Njegov skupni tok na začetku je enak 0. V prvi iteraciji najprej dodelimo nivoje vsem vozliščem v grafu z uporabo iskanja v širino, hkrati tudi preverimo, če je pretok še možen in če še obstaja pot od s do t v tem grafu.



Tukaj omenimo, da povezav, ki povezujejo vozlišči istega nivoja, ne moremo uporabiti v tej iteraciji. Na sliki vidimo, da ima vozlišče 1 dodeljen nivo 1, vozlišče 2 pa ima prav tako dodeljen nivo 1. Ker poteka povezava med vozliščema istega nivoja, je ne moremo uporabiti v tej iteraciji. Povezava

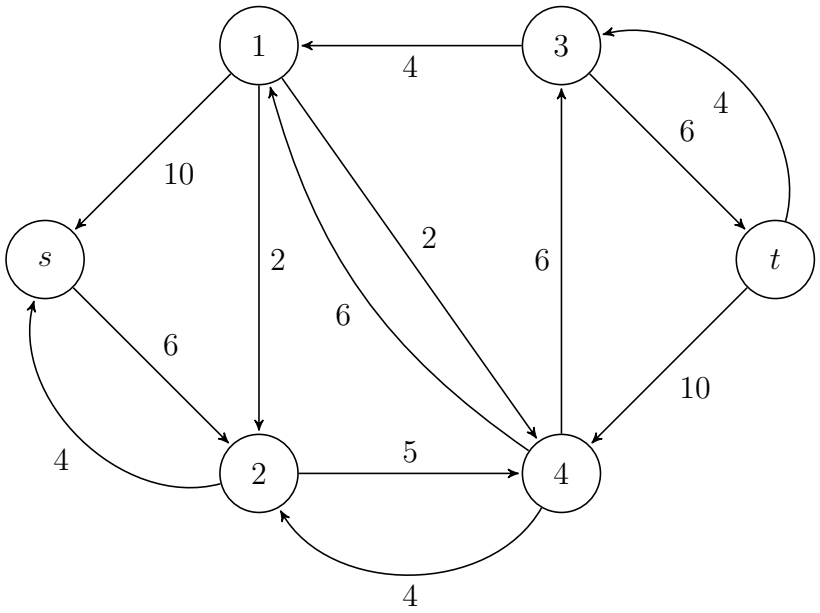
med vozliščem 1 nivoja 1 ter vozliščem 4 nivoja 2 je v tem primeru povsem veljavna in jo lahko uporabimo v tej iteraciji.

V naslednjem koraku najdemo blokiran pretok tako, da namesto vozlišč uporabimo vrednosti njihovih nivojev. Tu pošljemo tri tokove v isti iteraciji, kar predstavlja izboljšavo v primerjavi z Edmonds-Karp algoritmom, kjer smo lahko poslali le en tok.

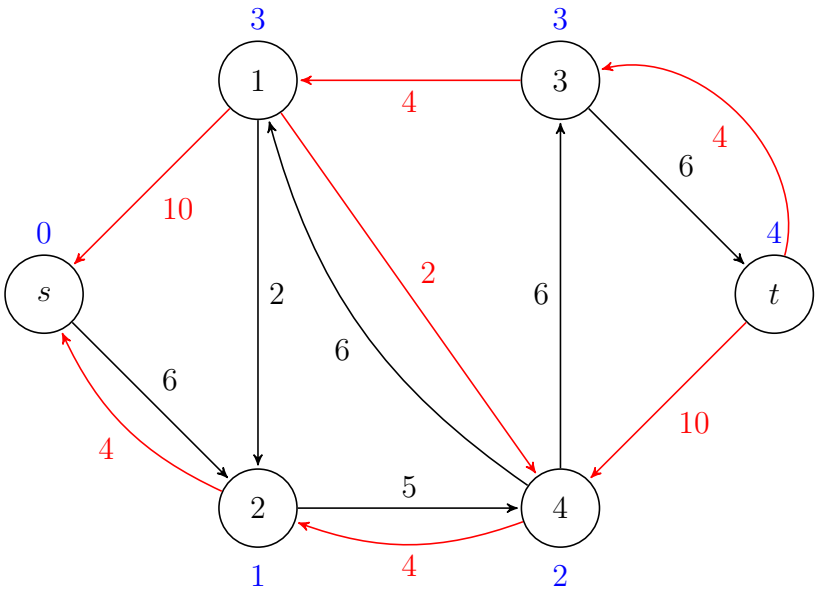
Tokovi so sledeči:

Tok	Enote toka	Pot
1	4	$s \to 1 \to 3 \to t$
2	6	$s \to 1 \to 4 \to t$
3	4	$s \to 2 \to 4 \to t$

Skupni tok je torej enak prejšnjemu skupnemu toku + vsoti vseh enot tokov v tej iteraciji, kar znaša 14 enot. Po eni iteraciji naš graf izgleda tako:

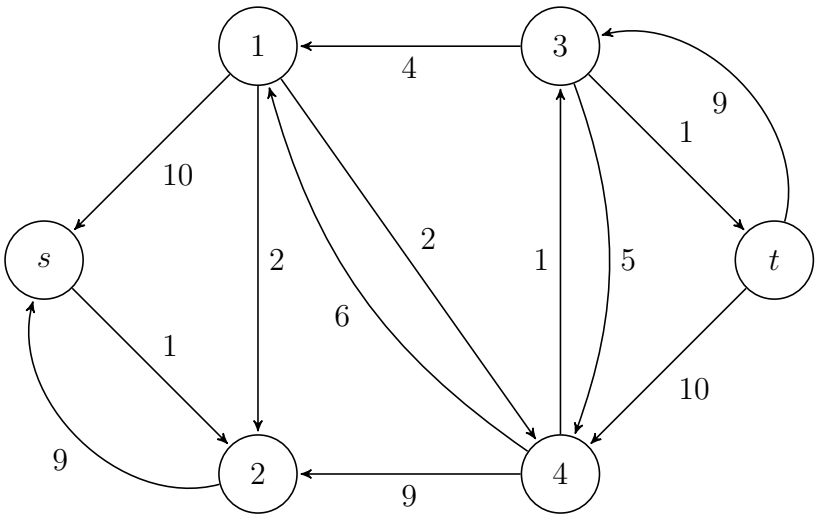


V drugi iteraciji ponovimo postopek. Dodelimo nivoje vsem vozliščem z uporabo iskanja v širino, prav tako preverimo, če obstajajo poti med izvorom in ponorom, preko katerih še lahko pošljemo tok.



Najdemo edini blokiran pretok v tej iteraciji, ki pošlje 5 enot toka na poti $s \to 2 \to 4 \to 3 \to t$. Skupni tok je enak prejšnjemu skupnemu toku, kateremu prištejemo še enote toka v tej iteraciji. Skupaj to znaša $14 + 5 = 19$ enot toka.

Nov graf izgleda tako:



Ker v tem koraku pot med s in t ne obstaja več, algoritem ustavimo. Končni maksimalen pretok, ki ga lahko pošljemo skozi graf, je torej 19.

5.2 Časovna zahtevnost

Časovna zahtevnost algoritma je enaka $O(n^2m)$, pri čemer je n število vozlišč, m pa število povezav v grafu. Uporaba iskanja v širino za konstruiranje nivojskega grafa namreč vzame $O(m)$ časa, pošiljanje več tokov, preden je pretok blokiran, pa porabi $O(nm)$ časa. Zunanja zanka se izvaja maksimalno $O(n)$ časa. V vsaki iteraciji skonstruiramo nov nivojski graf in najdemo blokiran pretok. Lahko dokažemo, da se število nivojev poveča vsaj za 1 v vsaki iteraciji, kar pomeni, da obstaja največ $n - 1$ blokiranih pretokov v algoritmu.

5.3 Dinamična drevesa

Čas izvajanja algoritma blokiranja pretoka je odvisen od sprememb tokov skozi povezave. Zato izboljšave tega algoritma iščemo z uporabo drugih podatkovnih struktur, ki omogočajo, da opravimo več takšnih sprememb v eni operaciji na podatkovni strukturi. To dosežemo z uporabo *dinamičnih dreves* [5,6], podatkovne strukture, ki si lahko zapomni nenasičene dele poti.

Povečevanje toka se opravlja s pomočjo operacij na dinamičnih drevesih, pri čemer takšna operacija stane logaritem dolžine poti, ki ji povečujemo tok. Takšna uporaba zmanjša čas izvajanja algoritma z $O(nm)$ na $O(n \log m)$.

Z omejitvijo maksimalne velikosti drevesa in z uporabo dodatnih podatkovnih struktur pa lahko dosežemo tudi časovno zahtevnost $O(nm \log (n^2/m))$.

Čeprav dinamična drevesa zagotavljajo najboljše pesimistične hitrosti, pa do sedaj niso bila uporabljena v praktičnih implementacijah, saj je večina praktičnih primerov relativno enostavnih, konstantni faktorji pri implementaciji dinamičnih dreves pa so zelo veliki.

Poglavje 6

Goldberg-Tarjan

Algoritem blokiranja pretoka uporablja globalne operacije, kot sta izgradnja pomožnega omrežja ter nasičenje na poti. *Metoda potiska in ponovnega označevanja* je dobila ime po dveh operacijah, ki jih izvaja lokalno, kar ji daje več fleksibilnosti in posledično možnost, da metodo pohitri v praksi. Metoda uporablja koncept *predtoka*, ki ga je predstavil Karzanov [8]. Njegov algoritem je prav tako uporabljal operacijo potiska, vendar pa je namesto ponovnega označevanja uporabljal razdalje v pomožnem omrežju, da je določil, kam naj se potisne tok. Metodo potiska in ponovnega označevanja sta predstavila Andrew V. Goldberg in Robert Tarjan [9]. Predstavila sta generično obliko algoritma s časovno zahtevnostjo $O(n^2m)$, sekvenčno implementacijo s časovno zahtevnostjo $O(n^3)$ ter implementacijo z uporabo dinamičnih dreves s časovno zahtevnostjo $O(nm \log (n^2/m))$.

Za lažjo vizualizacijo si povezave v grafu predstavljajmo kot cevi, vozlišča pa kot spoje. Izvor se nahaja na najvišjem nivoju in pošilja vodo vsem bližnjim vozliščem. Ko se v vozlišču nabere presežek vode, se ta potisne v vozlišče, ki je na nižjem nivoju. Če se voda ujame v določenem vozlišču (je ne moremo potisniti v nobeno drugo vozlišče), potem se vozlišče ponovno označi, kar pomeni, da se mu spremeni višina in se mu tako omogoči, da vodo zopet pošlje vozliščem na nižjem nivoju.

Vsako vozlišče ima pripadajoči spremenljivki, ki hranita vrednost trenu-

tne višine ter presežek toka. *Višina* se uporablja, da določimo, ali lahko pošljemo tok bližnjemu vozlišču ali ne. Tok se lahko pošlje le iz vozlišča z višjo višino v vozlišče z nižjo višino. *Presežek toka* je razlika toka, ki prihaja v vozlišče, in toka, ki odhaja iz njega. Vsaka povezava ima definiran tudi tok, ki trenutno poteka skozi njo, in pa kapaciteto.

6.1 Inicializacija predtoka

Ta operacija je zadolžena, da nastavi višino in tok vsakega vozlišča na 0. Višina izvora je odvisna od števila vozlišč v grafu. Nastavi se tudi tok vsake povezave na 0. Za vsako vozlišče, ki je sosednje izvoru, sta na začetku tok in presežek toka enaka kapaciteti.

6.2 Operacija potiska

Potisk se uporabi, da se pošlje presežek toka iz vozlišča. Če ima vozlišče presežek in obstaja bližnje vozlišče, ki ima nižjo višino, potem se presežek potisne tja. Količina potisnjenega toka skozi povezavo je enaka minimumu presežka in kapacitete povezave.

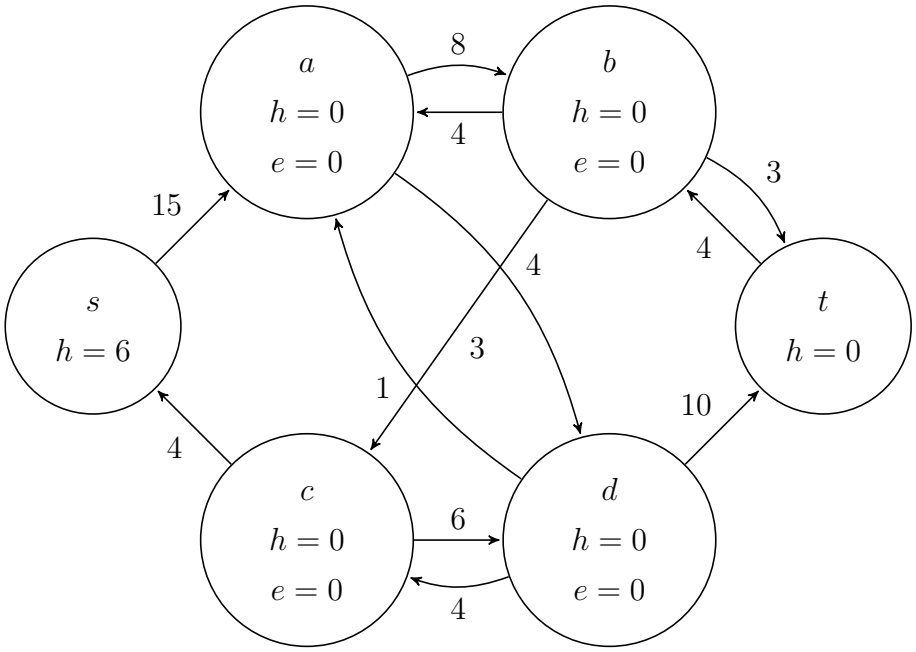
6.3 Operacija ponovnega označevanja

Ponovno označevanje se uporabi, ko ima vozlišče presežek in nobeno izmed bližnjih vozlišč nima nižje višine. V tem primeru spremenimo višino, da lahko izvedemo potisk. Novo višino nastavimo na vrednost višine bližnjega vozlišča, ki ima najmanjšo, še višjo višino in tej vrednosti prištejemo 1.

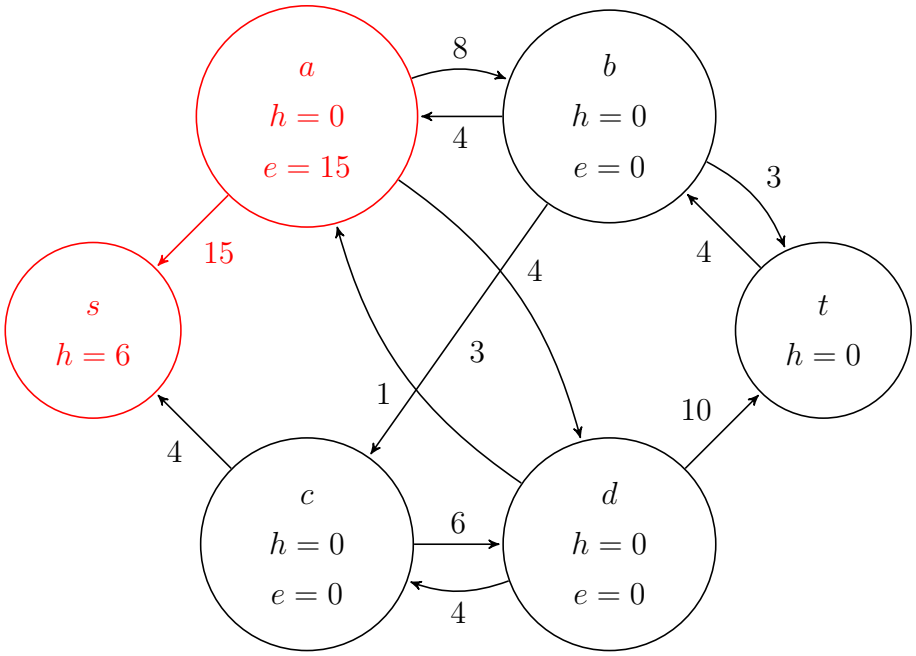
6.4 Primer izvajanja

Predstavili bomo primer izvajanja algoritma na primeru. Prvi korak je, da inicializiramo graf in nastavimo vrednosti predtoka na 0 ter inicializiramo

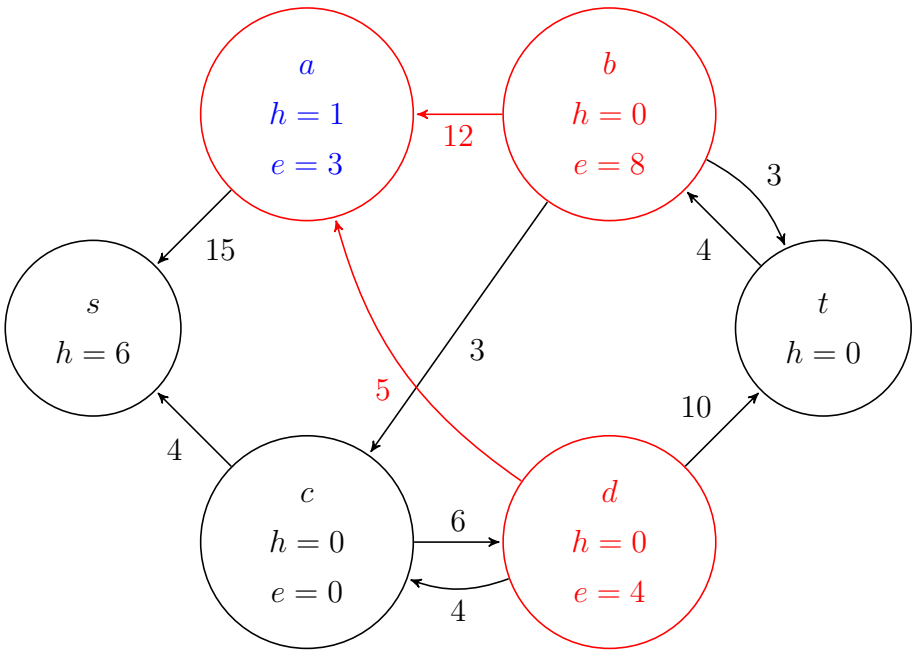
označevanje. V vozliščih spremenljivka h predstavlja višino, spremenljivka e pa presežek.



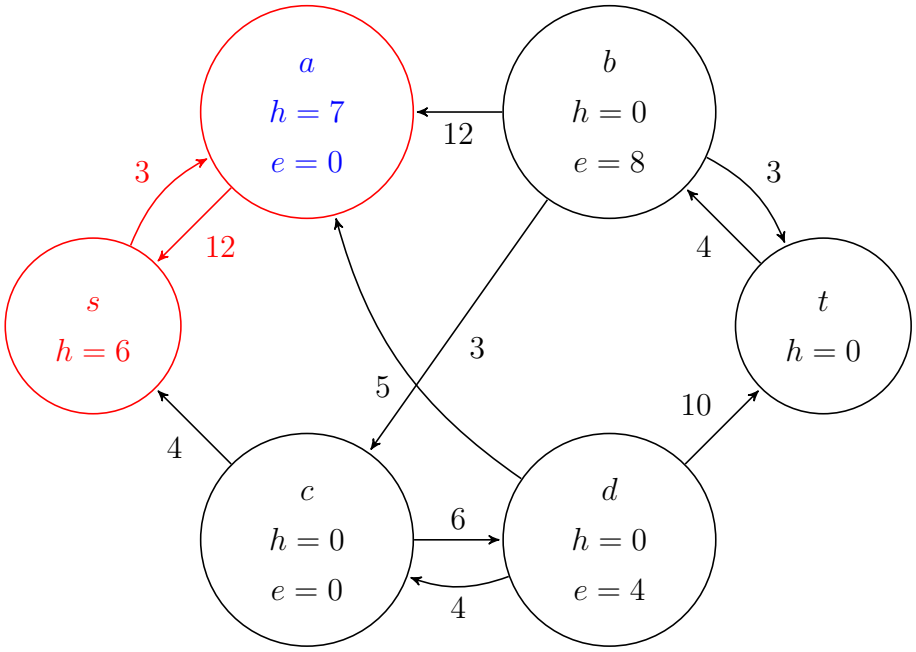
Začetni nasičevalni potisk se izvede nad vsemi povezavami predtoka, ki gredo iz izvora s.



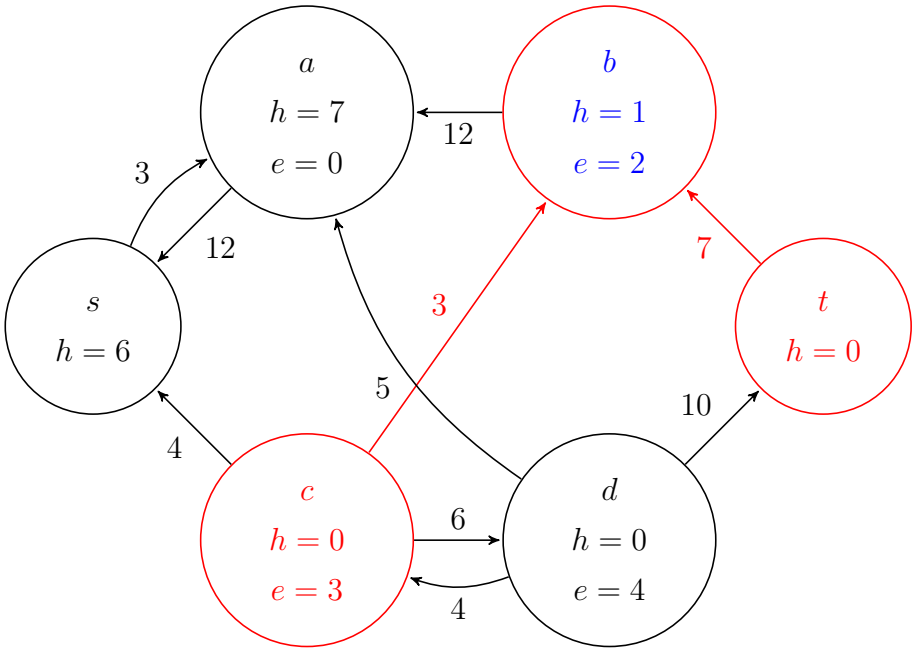
Vozlišče a se ponovno označi, da lahko potisnemo presežek toka proti ponoru t. Presežek v vozlišču a se nato potisne v vozlišče b in nato v d v dveh zaporednih nasičevalnih potiskih, kar pa še vedno pusti nekaj presežka v vozlišču a.



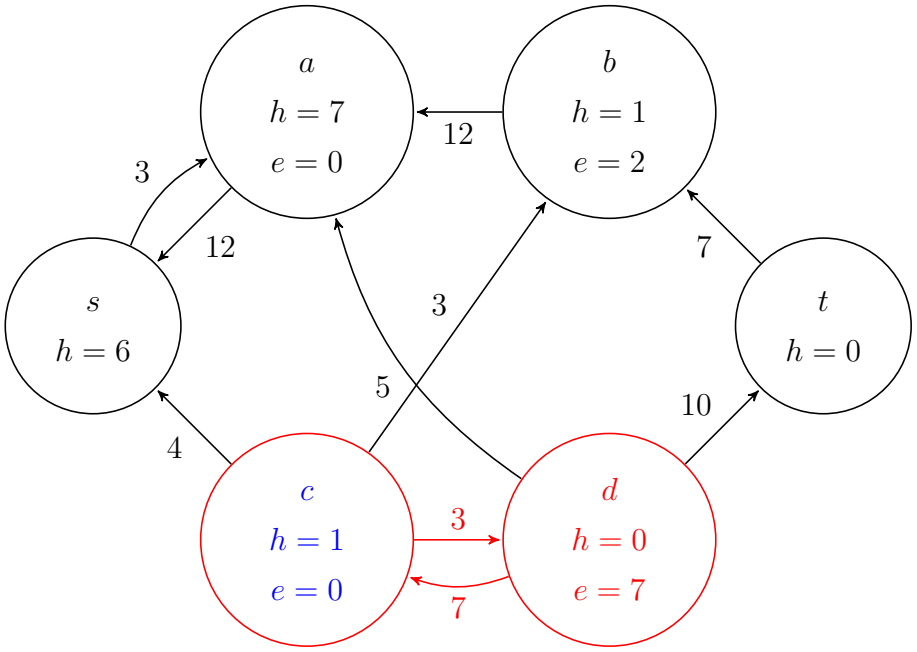
Vozlišče a je ponovno označeno, da potisne še preostanek svojega presežka nazaj v s. Vozlišče a je nato odstranjeno iz množice aktivnih vozlišč.



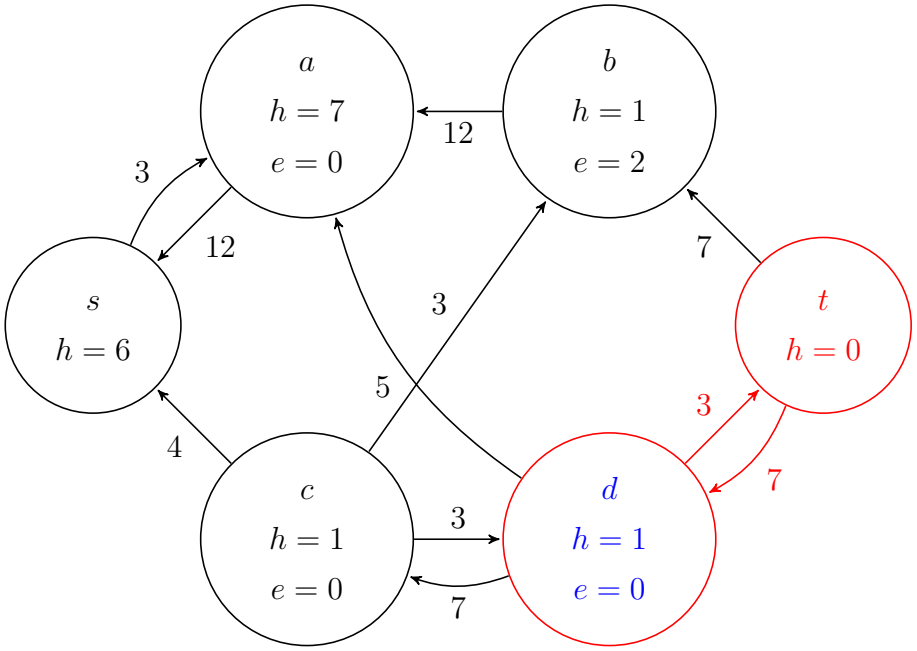
Ponovno se označi b, presežek pa se potisne do vozlišč t in d.



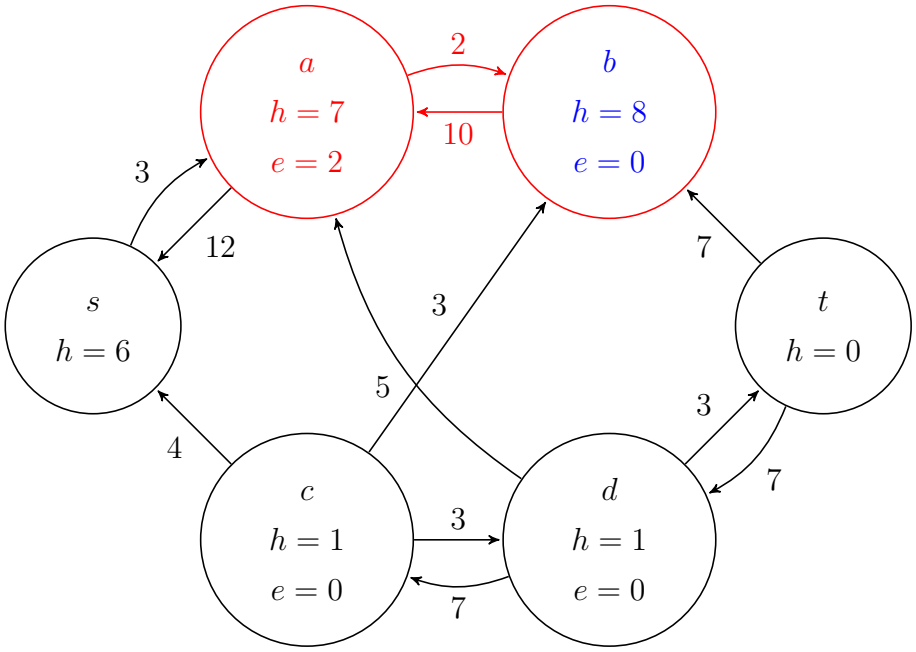
Ponovno se označi c, presežek pa se potisne do vozlišča d.



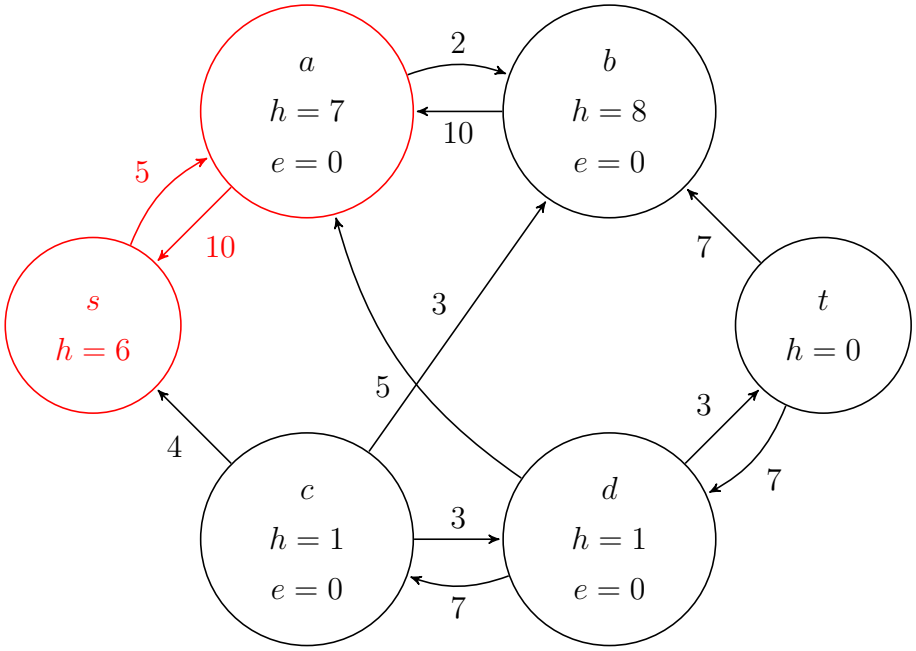
Ponovno se označi d, presežek pa se potisne do ponora t.



Vozlišče b tako ostane edino preostalo aktivno vozlišče, vendar ne more potisniti svojega presežka proti izvoru. Ponovno se označi b, presežek pa se nato potisne proti izvoru s preko vozlišča a.



Potisne se še preostanek presežka iz vozlišča a v izvor s. Ker sedaj ni več aktivnih vozlišč, se algoritem ustavi in vrne maksimalen pretok grafa, ki znaša 14.



6.5 Časovna zahtevnost

Časovna kompleksnost metode potiska in ponovnega označevanja je sledeča. Skupni čas ponovnega označevanje je $O(nm)$, čas nasičenih potiskov pa pravtako $O(nm)$. Čas, potreben za nenasičene potiske je $O(n^2m)$. Opis metode je sicer generičen, saj nismo definirali pravila za izbiro naslednjega aktivnega vozlišča, ki ga je potrebno procesirati. Nekateri drugi vrstni redi operacij namreč pripeljejo do boljših časov. Pri algoritmu potiska in ponovnega označevanja za najvišjo oznako, ki za vozlišče, ki bo procesirano naslednje, vedno izbere aktivno vozlišče z najvišjo oznako razdalje, velja, da sta čas, ki je potreben za nenasičene potiske, in tudi končna časovna zahtevnost enaka $O(n^2\sqrt{m})$. Z uporabo dinamičnih dreves pa lahko dobimo časovno zahtevnost $O(nm \log (n^2/m))$ enostavneje kot pa z uporabo metode blokiranja pretoka. Algoritem z najvišjo oznako je prav tako eden izmed najbolj praktičnih verzij metode. Seveda pa potrebujemo za robustno praktično uporabo dodatne hevristike. Metoda potiska in ponovnega označevanja je zelo prilagodljiva, saj omogoča enostavno dodajanje hevristik. Na primer, aktivna vozlišča bi lahko omejili le na te, ki imajo $d(v) < n$ in nato izvedli poprocesiranje (angl. postprocessing), da izračunamo končni tok. Lahko bi tudi naredili periodična vzvratna iskanja v širino za maksimiranje vrednosti $d(v)$. Primera izboljšav sta Cherkassky [10] in Goldberg [11].

Poglavje 7

Orlin

V članku iz leta 2012 [13] je James B. Orlin predstavil izboljšan algoritem za problem maksimalnega pretoka in dokazal, da se ga da rešiti v času $O(nm)$, kar je izboljšalo predhodno najboljši čas, ki so ga predstavili King, Rao in Tarjan [14].

7.1 Faze izboljšave

Recimo, da x predstavlja tok, $r = r[x]$ predstavlja vektor rezidualnih kapacitet, $s - t$ pa rez (S,T). Trojček (r, S, T) lahko uporabimo kot vhodno vrednost za izboljšavo. Fazo referenciramo kot Δ-*fazo izboljšave*, kjer Δ=$r(S, T)$. Torej, Δ je zgornja meja na maksimalnem rezidualnem pretoku med s in t. Izhodna vrednost je tok x', vektor $r' = r[x']$ ter $s - t$ rez (S', T'), da velja $r'(S, T) \leq \frac{\Delta}{(4m)}$. Fazo izboljšave lahko poženemo na grafu G ali pa na *kompaktnem omrežju*, ki je podatkovna struktura, ki jo je predstavil Orlin.

7.2 Obilni graf

Če (r, S, T) predstavlja vhodno vrednost faze izboljšave in če velja Δ=$r(S, T)$, potem je povezava (i, j) imenovana Δ-*obilna*, če velja $r_{ij} > 2\Delta$. Včasih jo imenujemo obilna tudi, če je to očitno že iz konteksta. Sprememba toka v

katerikoli povezavi je med fazo izboljšave največ Δ. Posledično velja, da če je povezava (i, j) Δ-obilna na začetku Δ-faze izboljšave, potem bo ostala obilna tudi v vseh kasnejših fazah.

Obilne povezave igrajo dve vlogi pri izboljšavi algoritma:

1. Usmerjeni cikli obsežnih povezav so sklenjeni v eno vozlišče. Sklenjene povezave se nato razširijo, ko algoritem določa optimalen tok v sklenjenem grafu.

2. Vozlišče se lahko stisne, če je vsaka povezava, ki si deli to vozlišče, obilna ali pa ima zelo nizko kapaciteto. Stisnjena vozlišča niso prisotna v stisnjenem omrežju.

Obilni graf je graf z množico vozlišč N, in množico obilnih povezav. Označimo ga kot G^{ab}. Obilni graf se skozi čas povečuje. Povezava (i, j) je v *prehodnem zaprtju* G^{ab}, če obstaja usmerjena pot v G^{ab} od vozlišča i do j. Algoritem ohranja prehodno zaprtje skozi vse iteracije. To lahko dosežemo v času $O(nm)$ z uporabo Italianovega [15] algoritma za dinamično ohranjanje prehodnega zaprtja grafa. Algoritem prehodnega zaprtja ohranja pot od i do j, če je ta pot obilna. Če obstaja več kot ena pot, potem ohranja prvo, ki jo določi. Poti ohranja implicitno z uporabo matrike M, kjer M_{ij} predstavlja vozlišče pred vozliščem j na poti od i do j v G^{ab}. Čas, ki je potreben za rekonstrukcijo poti P iz matrike M je $O(|P|)$. Algoritem prehodnega zaprtja je veljaven tudi, če G^{ab} vsebuje usmerjene cikle. Stiskanje obilnih grafov ne podaljša časa, potrebnega za ohranjanje dinamičnega prehodnega zaprtja.

7.3 Stiskanje

Če vsebuje obilni graf notranji povezavi (i, j) ter (j, i), potem lahko stisnemo vozlišči i in j v eno vozlišče in najdemo optimalen tok v *stisnjenem grafu*. Ko dobimo optimalen tok v stisnjenem grafu, lahko to vozlišče razširimo in dobimo začetni par povezav. Tok v *razširjenem grafu* se lahko izvede tako, da

pošljemo tok po (i, j) ali (j, i), odvisno od tega, kje je to potrebno, da lahko uravnavamo tok v vozliščih i in j. Skupni čas za stiskanje v fazi izboljševanja je $O(m)$, čas za razširitev stisnjenih ciklov pa prav tako $O(m)$. Več o stiskanju in razširjanju ciklov pa je mogoče prebrati v [16].

7.4 Časovna zahtevnost

Orlin v članku dokaže, da je za $m < n^{1.06}$ časovna zahtevnost algoritma $O(nm)$. Algoritem najde približno optimalen tok v fazi izboljševanja, in sicer z upoštevanjem treh različnih primerov. Predpostavimo, da c predstavlja število Δ-*kritičnih vozlišč*.

1. Če $c > m^{\frac{9}{16}}$, potem algoritem najde Δ'-optimalno rešitev, kjer $\Delta' = \frac{\Delta}{(4m)}$,
2. Če $m^{\frac{1}{3}} \leq c < m^{\frac{9}{16}}$, potem algoritem najde $\frac{\Delta'}{2}$ optimalno rešitev na G^c in pretvori v Δ'-optimalno rešitev na $G[r]$. Če $c < m^{\frac{1}{3}}$, potem najprej izbere parameter Γ, kjer $\Gamma < \Delta'$. Nato določi optimalen tok na (Δ, Γ)-kompaktnem omrežju in pretvori tok v Γ-optimalen tok na G.
3. Kreiranje kompaktnega omrežja vzame vsaj $m \log m$ korakov v vsaki fazi izboljšave. Zaradi tretjega primera je število faz $O(m^{\frac{2}{3}})$. To pove, da je skupni čas za kreiranje kompaktnega omrežja $O(m^{\frac{5}{3}} \log n)$ plus čas potreben za ohranjanje prehodnega zaprtja v obilnem grafu, ki znaša $O(nm)$.

Celoten dokaz izreka je na voljo v članku, tu smo predstavili le glavne ideje, ki vplivajo na izboljšavo.

Poglavje 8

Zaključek

8.1 Sklepne ugotovitve

Ugotovili smo, da je napredek pri razvoju algoritmov za iskanje maksimalnega pretoka potekal več kot pol stoletja. Problem, ki se ga je sprva dalo rešiti s splošnimi tehnikami linearnega programiranja, so številni reševali z novimi metodami, kjer so predstavili ideje, ki pospešijo čas izvajanja algoritma. Spoznali smo, da lahko že majhne spremembe, kot je vrstni red nenasičenih poti, ki jih izbiramo med izvajanjem, vplivajo na to, v kakšnem času se bo algoritem zaključil. Ugotovili smo tudi, da je Orlin predstavil trenutno najhitrejši algoritem, ki lahko problem reši v času $O(nm)$, kar je časovna zahtevnost, ki se jo je iskalo zelo dolgo. Razvoj nikakor ni končan, saj se še vedno izvajajo raziskave, ki bi predstavile algoritem, ki bi še pohitril iskanje maksimalnega pretoka. Prav tako so že zastavljene določene smernice, ki bi omogočile razvoj ne nujno najhitrejšega algoritma, pač pa algoritma, ki bi bil sila uporaben v praktičnih primerih.

8.2 Nadaljnje delo

To je tudi smer raziskave, ki je primerna za nadaljnje delo. Ker je področje zelo obširno, je razširitev možna v obliki magistrske naloge, kjer bi naredil

pregled še ostalih algoritmov, implementacijo vseh algoritmov, omenjenih v tej diplomski nalogi, in primerjavo z uporabo testov DIMACS, kjer bi bilo mogoče poiskati odgovor na to, kako se algoritmi obnašajo na različnih tipih grafov. V poštev pride tudi reševanje problema iz realnega sveta s pomočjo algoritmov za iskanje maksimalnega pretoka.

Literatura

[1] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford (2009). "26 Maximum Flow". Introduction to Algorithms, Third Edition. The MIT Press. 708–766.

[2] Ford, Jr., L.R. and Fulkerson D.R. Maximum flow through a network, Canadian Journal of Mathematics 8 (1956), 399-404.

[3] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford (2009). "29 Linear Programming". Introduction to Algorithms, Third Edition. The MIT Press. 843–897.

[4] Dantzig, G.B. Application of the simplex method to a transportation problem. In Activity Analysis and Production and Allocation, T.C. Koopmans, Ed. John Wiley and Sons, Inc., New York, 1951, 359–373.

[5] Sleator, D.D. and Tarjan, R.E. A data structure for dynamic trees. Journal of Computer and System Sciences 26, 3 (1983), 362–391.

[6] Sleator, D.D. and Tarjan, R.E. Self-adjusting binary search trees. Journal of the ACM 32, 3 (1985), 652–686.

[7] Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford (2009). "22.2 Breadth-first search". Introduction to Algorithms, Third Edition. The MIT Press. 594–600.

[8] Karzanov, A.V. Determining the maximal flow in a network by the method of preflows. Soviet Mathematical Dokladi 15 (1974), 434–437.

[9] Goldberg, A.V. and Tarjan, R.E. A new approach to the maximum flow problem. Journal of the ACM 35, 4 (1988), 921–940.

[10] Cherkassky, B.V. and Goldberg, A.V. On implementing push-relabel method for the maximum flow problem. Algorithmica 19, 4 (1997), 390–410.

[11] Goldberg, A.V. Two-level push-relabel algorithm for the maximum flow problem. In Proceedings of the Fifth Conference on Algorithmic Aspects in Information Management, Volume 5564 of Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2009, 212–225.

[12] Edmonds, J. and Karp, R.M. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19, 2 (1972), 248–264.

[13] Orlin, James B. Max Flows in O(Nm) Time, or Better. STOC '13, (2013), 765-774.

[14] V. King and S. Rao and R. Tarjan. A Faster Deterministic Maximum Flow Algorithm. Journal of Algorithms 17, 3 (1994), 447-474.

[15] G. Italiano. Amortized efficiency of a path retrieval data structure. Theoretical Computer Science, 48(0), (1986), 273-281.

[16] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. Journal of the ACM, 45, (1998) 783-797.

Stvarno kazalo