UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – 1. stopnja

Marcel Čampa

Algoritem potisni-povišaj za iskanje maksimalnih pretokov

Delo diplomskega seminarja

Mentor: prof. dr. Sergio Cabello

Ljubljana, 2017

Kazalo

1. Uvod 4
2. Osnovne definicije 4
3. Iskanje maksimalnega pretoka z algoritmom potisni-povišaj 4
3.1. O algoritmu 4
3.2. Primer delovanja algoritma 7
3.3. Implementacija algoritma v programskem jeziku Python in C++ 7
3.4. Pravilnosti delovanja algoritma 15
3.5. Časovna zahtevnost algoritma 18
Slovar strokovnih izrazov 19
Literatura 19

Algoritem potisni-povišaj za iskanje maksimalnih pretokov

Povzetek

Push-relabel algorithm for maximum flow problem

Abstract

Math. Subj. Class. (2010):
Ključne besede:
Keywords:

1. Uvod

2. Osnovne definicije

V tem razdelku se bomo spoznali z osnovnimi definicijami teorije grafov, brez katerih ne bomo mogli. Nato si bomo pogledali manj znane definicije in definicije specifične za algoritme tipa *potisni-povišaj*.

Definicija 2.1. Naj bo $G = (V, E, s, t)$ pretočno omrežje. Funkciji $f\colon V \times V \to \mathbb{N}_0$ pravimo **predtok**, če velja

(1) Za vsako povezavo $(u, v) \in E$ velja $f(u, v) \leq c(u, v)$.
(2) Za vsako vozlišče $u \in V$ velja $e(u) = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) \geq 0$.

Algoritmi tipa potisni-povišaj namreč ne ohranjajo Kirchoffovega zakona, ki velja za pretok. Zato pri algoritmih tega tipa govorimo o predtoku. Ker ne ohranjajo Kirchoffovih zakonov, definiramo naslednjo funkcijo.

Definicija 2.2. Naj bo $G = (V, E, s, t)$ pretočno omrežje. Funkciji $e\colon V \to \mathbb{N}_0$, ki je definirana kot

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

rečemo **presežek toka**. Vozlišču, za katerega velja $e(u) > 0$, pravimo **vozlišče v presežku**.

Z drugimi besedami bi lahko rekli, da funkcija e za vsako vozlišče pove, koliko preveč toka je vanj priteklo. To je ravno razlika med vsoto pritečenih tokov in vsoto odtečenih tokov.

Naslednja definicija nam bo dala nov atribut vozlišč.

Definicija 2.3. Naj bo $G = (V, E, s, t)$ pretočno omrežje. **Višinska funkcija** je funkcija $h\colon V \to \mathbb{N}_0$, za katero velja

(1) $h(s) = |V|$ in $h(t) = 0$,
(2) $h(u) \leq h(v) + 1$, za vsako povezavo $(u, v) \in E_f$.

3. Iskanje maksimalnega pretoka z algoritmom potisni-povišaj

V tem razdelku si bomo podrobneje ogledali algoritem *potisni-povišaj*. Začeli bomo s kratkim opisom delovanja algoritma in intuitivno razložili, kako algoritem deluje. Nato si bomo pogledali psevdokodo algoritma in se z njo pobližje spoznali na zgledu. Sledili bosta implementaciji algoritma v programskem jeziku Python in C++. Pokazali bomo pravilnost delovanja algoritma in njunih implementacij ter časovno zahtevnost algoritma. Na koncu razdelka pa si bomo vzeli še trenutek za primerjavo časov izvajanja obeh implementacij.

3.1. **O algoritmu.** Algoritem potisni-povišaj deluje po preprostem principu iz narave. Predsavljajmo si, da imamo rečno omrežje, ki se začne v eni točki in konča v eni točki. Z drugimi besedami imamo eno reko, ki pa se vmes poljubno deli in združuje. Seveda je na zemlji prisotna gravitacijska sila, ki povzroči, da voda teče od višje točke proti nižji, recimo od izvira v hribih do ponora v morje, vmes pa ubira tako pot, da nikjer ne gre navzgor. V jeziku grafov lahko predstavimo omenjeni pojav na naslednji način. Tam, kjer se reka deli oziroma združi, postavimo vozlišče

grafa. Del reke med dvema razvejiščema predstavlja povezavo med razvejiščema pripadajočima vozliščema. Izvir in ponor reke pa predstavljata vozlišči s in t. Vsakemu vozlišču pripišemo višino, na kateri se nahaja, in količino vode, ki je vanj pritekla in odtekla. Seveda se v naravi ne zgodi (razen v primeru neurij), da bi v razvejišče priteklo več vode, kot pa je je iz njega odteklo. Prav tako ne more priteči manj vode, kot je odteče.

Sedaj, ko smo se spomnili, kako deluje mati narava, in to prevedli v matematični jezik, si podrobneje poglejmo, kako deluje algoritem potisni-povišaj. Začnemo z omrežjem (od sedaj bomo rajši kot o grafih govorili o omrežjih) $G = (V, E, s, t)$ in funkcijama $c\colon V \times V \to \mathbb{N}$, ki vsaki povezavi priredi njeno kapaciteto, in $f\colon V \times V \to \mathbb{N}$, ki za vsako povezavo pove, koliko vode teče v nekem trenutku preko nje. Vozliščem $v \in V$ priredimo še funkciji $h\colon V \to \mathbb{N}_0$, ki določa višino vozlišča, in $e\colon V \to \mathbb{N}_0$, ki pove, koliko preveč vode je priteklo v neko vozlišče. Seveda velja

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

Algoritem na začetku nastavi višino vseh vozlišč razen vozlišča s na nič in višino s na $|V|$. Tako na začetku velja $h(s) = |V|$ in $h(u) = 0$, $u \in V \setminus s$. Nato potisnemo iz s tok v sosednja vozlišča tako, da zasičimo povezave, torej da velja $f(s, v) = c(s, v)$, za vse $v \in V$, za katere velja $(s, v) \in E$. Poleg tega dodamo v residualno omrežje še obratne povezave, katerim nastavimo $f(v, s) = -f(s, v)$, da zadostimo pogoju (manjka referenca na definicijo toka). S tem smo ustvarili tako imenovani *predtok*. To smo lahko storili, ker je višina vozlišča s večja kot višina sosednjih vozlišč v, saj $h(s) = |V| > 0 = h(v)$.

Rezultat te operacije je, da se je v vozliščih, sosednjih vozlišču s, nabrala odvečna voda. Za ta vozlišča torej velja $e(v) > 0$. Sedaj lahko potisnemo vodo iz teh vozlišč naprej, saj je vode v njih preveč, želimo pa, da je odteče toliko, kot je priteče. Vendar tega ne moremo storiti, saj so višine sosednjih vozlišč prav tako enake nič. Zato si izberemo neko vozlišče u, v katerega je priteklo preveč vode, in mu povečamo višino na $\min\{h(v) : (u, v) \in E_f\} + 1$. S tem smo omogočili, da bo voda odtekla v vsaj eno izmed vozlišč. Ta postopek ponavljamo, dokler lahko potisnemo tok v omrežju ali pa povišamo neko vozlišče. Tok, ki na koncu priteče v t, je enak maksimalnemu pretoku omrežja, kar bomo pokazali kasneje.

Oglejmo si sedaj psevdokodo algoritma. Spoznali smo, da je algoritem sestavljen iz dveh osnovnih operacij, *potiskanja* in *povišanja*, zato se posvetimo tema operacijama. Začnimo s potiskanjem.

```
POTISNI (u, v)
1   // Potisnemo lahko, če je e(u) > 0
2   // c(u,v) > 0 in h(u) = h(v) + 1.
3   delta = min{ e(u), c(u,v) - f(u,v) }
4   ČE (u,v) v E, POTEM
5       f(u,v) += delta
6   DRUGAČE f(v,u) -= delta
7   e(u) -= delta
8   e(v) += delta
```

To operacijo lahko storimo, če ima u presežek toka, torej, če velja $e(u) > 0$, če je kapaciteta povezave $c(u, v) > 0$ in če sta vozlišči u in v na primernih višinah, torej če velja $h(u) = h(v) + 1$. Najprej izračunamo, kolikšno količino Δ lahko potisnemo. Ta je enaka minimumu med presežkom toka v vozlišču u in residualno kapaciteto povezave, ki je enaka $c(u, v) - f(u, v)$. To storimo v vrstici 2. V vrsticah 3–6 potisnemp tok Δ po povezavi (u, v), če ta povezava obstaja. V nasprotnem primeru potisnemo $-\Delta$ po obratni (residualni) povezavi. V vrsticah 6 in 7 posodobimo še presežek toka v krajiščih povezave. To storimo tako, da v začetnem vozlišču presežek zmanjšamo za tok, ki smo ga potisnili, v končnem vozlišču pa presežek povečamo.

Nadaljujmo z operacijo povišanja vozlišča.

```
POVIŠAJ (u)
1   // Vozlišče u povišamo, če je e(u) > 0 in
2   // za vsak v iz V, (u,v) v E_f, velja h(u) <= h(v).
3   h(u) = min{h(v) : (u,v) v E_f} + 1
```

Operacija povišanja vozlišča u je precej enostavna. Storimo jo takrat, ko ima vozlišče u presežek toka, torej velja $e(u) > 0$, a hkrati ne moremo potisniti toka v sosednja vozlišča, saj so vsa na večji ali enaki višini kot u.

V opisu algoritma smo navedli še *inicializacijo predtoka*. Zapišimo psevdokodo za to operacijo.

```
INICIALIZIRAJ_PREDTOK(G,s)
 1   // V grafu G si izberemo vozlišče s in inicializiramo predtok.
 2   ZA vsak v v V(G)
 3       h(v) = 0
 4       e(v) = 0
 5   ZA vsak (u,v) v E(G)
 6       f(u,v) = 0
 7   h(s) = |V|
 8   ZA vsak v, za katerega obstaja (s,v) v E(G)
 9       f(s,v) = c(s,v)
10       e(v) = f(s,v)
```

Kot smo dejali, zgornja operacija nastavi višine vozlišč in presežek toka v vozliščih na nič. To storimo v vrsticah 2–6. V vrstici 7 nato nastavimo višino vozlišča s na število vseh vozlišč v omrežju, torej $h(s) = |V|$. V vrsticah 8–10 nato potisnemo tok prek vseh povezav, ki izhajajo iz s in v vrstici 10 popravimo še presežek toka v vozlišču s sosednjih vozliščih. Opazimo, da nismo odšteli presežka toka v vozlišču na začetku povezave, torej v vozlišču s. Tega nismo storili, ker je to nepotrebno; predstavljamo si namreč, da je v vozlišču s lahko poljubna količina vode, več kot je potrebujemo, več je lahko dobimo. Kasneje bomo videli, kaj se zgodi, če smo v inicializaciji predtoka poslali preveč vode, kot je omrežje lahko spusti skozi.

Čas je, da navedemo še glavni del algoritma, torej ,,program", ki uporablja zgoraj navedene operacije. Psevdokoda je na videz precej preprosta.

```
POTISNI-POVIŠAJ(G,s)
1   INICIALIZIRAJ_PREDTOK(G,s)
```

```
2   DOKLER obstaja mogoča operacija POTISNI ali POVIŠAJ
3       izvedi mogočo operacijo
```

Na videz nedolžna, vendar skriva rahlo prepreko do povsem direktne implementacije. Vprašanje, ki se pojavi, je namreč, kako vedeti, ali lahko potisnemo in preimenujemo. Oglejmo si zgled delovanja algoritma in sproti se nam morda porodi ideja.

3.2. **Primer delovanja algoritma.** Vzemimo preprosto omrežje na sedmih vozliščih. Naj velja $G = (V, E, s, t)$, kjer je

$$V = \{0, 1, 2, 3, 4, 5, 6\},$$
$$E = \{(0,1), (0,2), (0,3), (1,3), (1,5), (2,4), (3,4), (3,6), (4,6), (5,6)\},$$
$$s = 0,$$
$$t = 6.$$

Kapacitete vozlišč so podane v naslednji tabeli.

Tabela 1. Kapacitete povezav omrežja G.

	$(0,1)$	$(0,2)$	$(0,3)$	$(1,3)$	$(1,5)$	$(2,4)$	$(3,4)$	$(3,6)$	$(4,6)$	$(5,6)$
$c(u,v)$	10	3	7	8	5	4	3	12	2	4

Omrežje G na začetku izgleda takole:

Poiščimo sedaj maksimalni pretok skozi to omrežje s pomočjo zgoraj opisanega algoritma potisni-povišaj.

3.3. **Implementacija algoritma v programskem jeziku Python in C++.** Najprej si oglejmo idejo implementacije v Pythonu. Ker iščemo maksimalni pretok v grafu, si definiramo razred `Graf`, ki vsebuje dva podrazreda, to sta podrazred `Vozlisce` in podrazred `Povezava`, ki predstavljata očitno.

Za vsako vozlišče $u \in V$ si moramo zapomniti njegovo višino $h(u)$ in presežek toka v vozlišču $e(u)$, zato podrazred `Vozlisce` vsebuje dve vrednosti, `Vozlisce.h`, ki predstavlja višino vozlišča, in `Vozlisce.e`, ki predstavlja presežek toka.

Podobno si moramo za vsako povezavo $(u, v) \in E$ zapomniti, kje se začne, torej u, in kje konča, torej v, ter njeno kapaciteto $c(u, v)$ in trenuten tok čeznjo $f(u, v)$. Tako podrazred `Povezava` vsebuje štiri vrednosti: `Povezava.u` in `Povezava.v` vsebujeta začetno in končno vozlišče povezave, `Povezava.c` vsebuje podatek o kapaciteti povezave in `Povezava.f` trenuten tok preko povezave.

Ker graf sestoji iz vozlišč in povezav, vsebuje razred `Graf` dva seznama – to sta `Graf.vozlisca` in `Graf.povezave` –. V razredu `Graf` nato definiramo metode, ki nam izračunajo maksimalni pretok skozi graf. Navedimo za začetek le kratke opise metod, podrobnejše si bomo pogledali pri implementaciji v C++, saj bo tam vse skupaj lažje berljivo.

- `maksimalni_pretok()`: Izračuna maksimalni pretok v grafu.
- `inicializiraj_predtok()`: Inicializira predtok.
- `potisni(u)`: Poskusi opraviti operacijo potiska iz vozlišča u in nas obvesti, ali ji je uspelo ali ne.

- `povisaj(u)`: Poveča višino vozlišča u.
- `vozlisce_s_presezkom()`: Poišče vozlišče, v katerem je presežek toka.
- `posodobi_obratno_povezavo(i, delta)`: Od obratne povezave odšteje tok, ki smo go ravno poslali. Če povezave ne najde, jo doda v residualnem grafu.
- `dodaj_povezavo(u,v,c,f)`: V graf doda povezavo (u, v) in pripadojočo kapaciteto c ter tok čez povezavo f.

Oglejmo si sedaj implementacijo.

```
# Implementacija algoritma potisni-povisaj v Pythonu

class Graf:
    class Vozlisce:
        def __init__(self, h, e):
            self.h = h
            self.e = e

    class Povezava:
        def __init__(self, u, v, c, f):
            self.u = u
            self.v = v
            self.c = c
            self.f = f

    def __init__(self, V):
        self.vozlisca = [self.Vozlisce(0, 0) for _ in range(V)]
        self.povezave = []

    def maksimalni_pretok(self):
        self.inicializiraj_predtok()

        while self.vozlisce_s_presezkom() != -1:
            u = self.vozlisce_s_presezkom()
            if not self.potisni(u):
                self.povisaj(u)

        return self.vozlisca[len(self.vozlisca)-1].e

    def inicializiraj_predtok(self):
        self.vozlisca[0].h = len(self.vozlisca)

        for i in range(len(self.povezave)):
            if self.povezave[i].u == 0:
                self.povezave[i].f = self.povezave[i].c
                self.vozlisca[self.povezave[i].v].e =\
                    self.povezave[i].f
                self.posodobi_obratno_povezavo(i, self.povezave[i].f)

    def potisni(self, u):
```

```
        for i in range(len(self.povezave)):
            if self.povezave[i].u == u and\
                    self.povezave[i].f < self.povezave[i].c and\
                    self.vozlisca[u].h  == \
                    self.vozlisca[self.povezave[i].v].h + 1:
                delta = min(self.vozlisca[u].e,\
                    self.povezave[i].c - self.povezave[i].f)
                self.vozlisca[u].e -= delta
                self.vozlisca[self.povezave[i].v].e += delta
                self.povezave[i].f += delta

                self.posodobi_obratno_povezavo(i, delta)
                return 1
        return 0

    def povisaj(self, u):
        min_v = sys.maxint

        for i in range(len(self.povezave)):
            if self.povezave[i].u == u and\
                    self.povezave[i].f < self.povezave[i].c and\
                    self.vozlisca[self.povezave[i].v].h < min_v:
                min_v = self.vozlisca[self.povezave[i].v].h

        self.vozlisca[u].h = min_v + 1;

    def vozlisce_s_presezkom(self):
        for u in range(1, len(self.vozlisca)-1):
            if self.vozlisca[u].e > 0:
                return u
        return -1

    def posodobi_obratno_povezavo(self, i, delta):
        for j in range(len(self.povezave)):
            if self.povezave[i].u == self.povezave[j].v and\
                    self.povezave[i].v == self.povezave[j].u:
                self.povezave[j].f -= delta
                return
        self.dodaj_povezavo(\
            self.povezave[i].v, self.povezave[i].u, 0, -delta)

    def dodaj_povezavo(self, u, v, c, f):
        self.povezave.append(self.Povezava(u, v, c, f))


def napolni_graf():
    V = int(raw_input())
    G = Graf(V)
```

```
    while True:
        try:
            u, v, c = raw_input().split()
            G.dodaj_povezavo(int(u), int(v), int(c), 0)
        except EOFError:
            break

    return G


if __name__ == '__main__':
    import sys

    G = napolni_graf()
    print "Maksimalni pretok je {0}.".format(G.maksimalni_pretok())
```

Opazimo dve naslednji dve stvari. Najpomembnejša stvar, ki jo opazimo je, da smo v metodi `makimalni_pretok` zahtevali, da se program izvaja dokler obstaja vozlišče s presežkom. Spomnimo se, da smo v psevdokodi namreč zapisali, da se mora program izvajati dokler obstaja mogoča operacija potiska ali povišanja. Da sta stvari ekvivalentni, si bomo pogledali v dokazu pravilnosti algoritma (lema 3.2).

Druga stvar pa je, da metoda `potisni(`u`)` sprejme le en argument, medtem ko smo v psevdokodi definirali operacijo kot `POTISNI(`u`,`v`)`. Razlog je v tem, da smo zaradi lažje implementacije prestavili pregledovanje, v katere sosede lahko potisnemo tok, znotraj metode `potisni(`u`)`, zato ne rabimo dveh vozlišč kot argumenta. S tem imata seveda operacija `POTISNI(`u`,`v`)` in metoda `POTISNI(`u`)` precej drugačno časovno zahtevnost.

V programskem jeziku C++ je stvar skoraj povsem identična. Tu smo se le izognili razredom in podrazredom. Razred `Graf` smo povsem opustili in informacije o vozliščih in povezavah grafa shranili kar v globalni spremenljivki. Podrazreda `Vozlisce` in `Povezava` smo nadomestili s strukturama. Program je podrobneje pokomentiran, tako da na tem mestu ni smiselno navajati, kako deluje.

```
// Implementacija algoritma potisni-povisaj v C++.

//=================================================================
//        KNJIZNICE IN DEFINICIJE
//=================================================================

#include<iostream>
#include<vector>
#include<climits>
#include<cstdio>

using namespace std;

// Vozlisce in povezavo predstavimo s strukturo.
struct Vozlisce
```

```
{
    int h, e;

    Vozlisce(int h, int e)
    {
        this->h = h;
        this->e = e;
    }
};

struct Povezava
{
    int u, v;
    int c;
    int f;

    Povezava(int u, int v, int c, int f)
    {
        this->u = u;
        this->v = v;
        this->c = c;
        this->f = f;
    }
};

// Prototipi funkcij.
int potisni_povisaj();
void inicializiraj_predtok();
int potisni(int u);
void povisaj(int u);
int vozlisce_s_presezkom();
void posodobi_obratno_povezavo(int i, int delta);
void napolni_graf();

// Globalne spremenljivke.
int V; // Stevilo vozlisc. Sicer velja V = vozlisca.size().
vector<Vozlisce> vozlisca;
vector<Povezava> povezave;

//=====================================================================
//        MAIN
//=====================================================================

// Main funkcija, uporabljena za testiranje programa.
int main()
{
    // S standardnega vhoda preberi podatke o vozliscih
    // in povezavah in napolni vektorja vozlisca in povezave.
    napolni_graf();
```

```
    // Izvedi algoritem potisni-povisaj in izpisi rezultat.
    cout << "Maksimalni pretok je " << potisni_povisaj() << endl;
}

//=====================================================================
//        ALGORITEM
//=====================================================================

// Glavna funkcija algoritma potisni-povisaj. Najprej
// inicializira predtok, potem pa izvaja operacije potiska
// in povisanja, kakor je pac potrebno. To pocne, dokler
// obstaja vozlisce s presezkom (to ne moreta biti s in t).
int potisni_povisaj()
{
    inicializiraj_predtok();

    while (vozlisce_s_presezkom() != -1)
    {
        int u = vozlisce_s_presezkom();

        // Ce ne mores potisniti, potem povecaj visino vozlisca.
        if (!potisni(u))
            povisaj(u);
    }

    // Vrni presezek v zadnjem vozliscu, to je ravno t.
    return vozlisca[V-1].e;
}

// Inicializira predtok. To pomeni, da nastavi visino vozlisca s
// in potisne tok iz s v vsa sosednja vozlisca. Pri tem zasici
// povezave.
void inicializiraj_predtok()
{
    // Visina vozlisca s je enaka stevilu vozlisc.
    vozlisca[0].h = V;

    for (int i = 0; i < povezave.size(); i++)
    {
        // Ce se povezava zacne v vozliscu s...
        if (povezave[i].u == 0)
        {
            // Zasici povezavo.
            povezave[i].f = povezave[i].c;

            // Sosednje vozlisce dobi presezek toka.
            vozlisca[povezave[i].v].e = povezave[i].f;
```

```
            // Posodobimo obratno povezavo. Ce je ni,
            // se doda nova v residualnem grafu.
            posodobi_obratno_povezavo(i, povezave[i].f);
        }
    }
}

// Operacija potisni. Najprej preverimo, ali se povezava zacne
// v vozliscu u, ce je na povezavi se kaj residualne kapacitete,
// torej ali je povezava se nezasicena. Ce je vozlisce u na
// vecji visini kot sosed, potem izvrsi potisk.
int potisni(int u)
{
    for (int i = 0; i < povezave.size(); i++)
    {
        // Prepricaj se, da pogoji drzijo.
        if (povezave[i].u == u && povezave[i].f < povezave[i].c &&
                vozlisca[u].h == vozlisca[povezave[i].v].h + 1)
        {
            // Potisnemo lahko manjse izmed presezka vozlisca u
            // ter residualne kapacitete povezave.
            int delta =
                min(vozlisca[u].e, povezave[i].c - povezave[i].f);

            // Presezek u je pomanjsan za toliko,
            // kolikor smo potisnili.
            vozlisca[u].e -= delta;

            // V sosedu se za ravno toliko poveca presezek.
            vozlisca[povezave[i].v].e += delta;

            // Tok cez povezavo se poveca.
            povezave[i].f += delta;

            // Posodobimo tok po obratni povezavi. Ce povezave
            // ni, jo dodamo v residualnem grafu.
            posodobi_obratno_povezavo(i, delta);

            // Potisk uspesen.
            return 1;
        }
    }

    // Nismo uspeli najti povezavi, po kateri bi lahko izvrsili
    // potisk. To pomeni, da bo vozliscu potrebno povecati visino.
    return 0;
}

// Operacija povisaj. Nastavi visino vozlisca u na minimum visin
```

```
// sosednjih vozlisc + 1.
void povisaj(int u)
{
    // Zacetna minimalna visina naj bo nekaj velikega.
    int min_visina_sosedov = INT_MAX;

    for (int i = 0; i < povezave.size(); i++)
    {
        // Prepricati se moramo, da se povezava zacne v u
        // in da ni nasicena. Ce velja se, da je visina
        // drugega konca povezave (torej soseda od u) manjsa
        // kot trenutna najmanjsa, posodobimo trenutno najmanjso.
        if (povezave[i].u == u && povezave[i].f < povezave[i].c &&
                vozlisca[povezave[i].v].h < min_visina_sosedov)
            min_visina_sosedov = vozlisca[povezave[i].v].h;
    }

    // Spremenimo visino vozlisca u na 1 + minimalna visina sosedov.
    vozlisca[u].h = min_visina_sosedov + 1;
}

//======================================================================
//        POMOZNE FUNKCIJE
//======================================================================

// Poisce vozlisce s presezkom in vrne njegov indeks.
// Ce vozlisca ne najde, vrne sentinel.
int vozlisce_s_presezkom()
{
    for (int u = 1; u < V-1; u++)
    {
        if (vozlisca[u].e > 0)
            return u;
    }

    return -1;
}

// Posodobi tok na obratni povezavi (povezavi, ki ima
// konca v istih vozliscih, kakor tista, po kateri smo
// potisnili tok, in kaze v drugo smer). Ce te povezave
// ni, jo dodaj v residualnem grafu.
void posodobi_obratno_povezavo(int i, int delta)
{
    for (int j = 0; j < povezave.size(); j++)
    {
        // Prepricamo se, da gre za obratno povezavo.
        if (povezave[i].u == povezave[j].v &&
                povezave[i].v == povezave[j].u)
```

```
        {
            // Odstej tok, ki je bil potisnjen in koncaj.
            povezave[j].f -= delta;
            return;
        }
    }

    // Ni bilo obratne povezave. To pomeni, da jo moramo
    // dodati v residualnem grafu.
    povezave.push_back(
        Povezava(povezave[i].v, povezave[i].u, 0, -delta));
}


// S standardnega vhoda prebere podatke o grafu
// in napolni vektorja vozlisc in povezav.
void napolni_graf()
{
    scanf("%d\n", &V);
    for (int i = 0; i < V; i++)
        vozlisca.push_back(Vozlisce(0, 0));

    int u, v, c;
    while (scanf("%d %d %d\n", &u, &v, &c) != EOF)
        povezave.push_back(Povezava(u, v, c, 0));
}

//============================================================================
```

3.4. **Pravilnosti delovanja algoritma.** V tem podrazdelku bomo s pomočjo lem pokazali, da tako algoritem `POTISNI-POVIŠAJ` kot tudi njegova implementacija delujeta pravilno. S tem mislimo na to, da se algoritem konča in ob tem vrne pravilen rezultat, torej res pretok, ki je maksimalen.

Če si pogledamo, kako deluje operacija potisni, vidimo, da nikjer v kodi ne uporabimo dejstva, da je razlika višine med vozliščema nujno ena. Vendar to še vseeno zahtevamo.

Lema 3.1. *Naj bo $G = (V, E)$ pretočno omrežje, $f\colon V \times V \to \mathbb{N}_0$ predtok v G in $h\colon V \to \mathbb{N}_0$ višinska funkcija. Potem za vsaki vozlišči $u, v \in V$ velja, da če je $h(u) > h(v) + 1$, potem povezava (u, v) ni v residualnem omrežju.*

Ta lema nam pove, da ne obstaja residualna povezava med u in v, če je $h(u) > h(v) + 1$. To pomeni, da če potisnemo tok v vozlišče za več kot ena nižje, ne bomo naredili nič konkretnega, kar se je preprosto prepričati.

Sedaj si poglejmo obljubljeno lemo, ki nam zagotavlja pravilnost delovanja implementacije. Spomnimo se namreč, da smo v implementaciji algoritem opravljali, dokler je bilo kakšno vozlišče s presežkom toka, čeprav smo v psevdokodi zapisali,

da moramo algoritem opravljati, dokler je mogoča katera izmed operacij `POTISNI` in `POVIŠAJ`.

Lema 3.2 (na vozlišču s presežkom lahko opravimo ali potisk ali povišanje)**.** *Naj bo $G = (V, E, s, t)$ pretočno omrežje, f predtok, h višinska funkcija in $e \times V \to \mathbb{N}_0$ funkcija, ki za vsako vozlišče pove, kolikšen je v njem presežek toka. Če ima vozlišče $u \in V$ presežek toka, torej $e(u) > 0$, potem lahko na tem vozlišču opravimo ali operacijo potisni ali operacijo povišaj.*

Dokaz. Naj ima u presežek toka. Za vsako residualno povezavo (u, v) velja $h(u) \leq h(v) + 1$, ker je h višinska funkcija. Če ne moremo opraviti operacije potisni, potem za vse residualne povezave (u, v) velja $h(u) < h(v) + 1$, oziroma $h(u) \leq h(v)$. Torej lahko opravimo operacijo povišanja. □

Oglejmo si tri leme o višinski funkciji.

Lema 3.3 (višine vozlišč se nikoli ne zmanjšajo)**.** *Med izvajanjem programa `POTISNI-POVIŠAJ` velja za vsako vozlišče $u \in V$, da se $h(u)$ nikoli ne zmanjša. Še več, vsakič, ko na u opravimo povišanje, se njegova višina poveča za vsaj ena.*

Dokaz. Ker se višine vozlišč spreminjajo le med povišanji, je za dokaz celotne leme zadosti pokazati drugi del leme. Naj bo sedaj u vozlišče, na katerem opravljamo povišanje. Torej za vse $v \in V$, za katere je $(u, v) \in E_f$, velja $h(u) \leq h(v)$. Ker to velja za vsak v, velja tudi

$$u \leq \min_{(u,v) \in E_f} h(v)$$

kar pa je ekvivalentno

$$u < 1 + \min_{(u,v) \in E_f} h(v).$$

□

Lema 3.4. *Med izvajanjem programa `POTISNI-POVIŠAJ(G,s)` h vedno zadrži lastnosti višinske funkcije, opisane v definiciji 2.3.*

Dokaz. Dokaz bomo naredili s pomočjo indukcije na število osnovnih operacij.

Po inicializaciji predtoka je h očitno višinska funkcija.

Poglejmo si najprej, kaj se zgodi med operacijo `POVIŠAJ(`u`)`. `POVIŠAJ(`u`)` zagotovi, da za vsako residualno povezavo $(u, v) \in E_f$ po opravljeni operaciji velja $h(u) \leq h(v) + 1$. Vzemimo sedaj residualno povezavo, ki vstopa v u, recimo $(w, u) \in E_f$. Po lemi 3.3 iz $h(w) \leq h(u) + 1$ pred operacijo sledi $h(w) < h(u) + 1$ po operaciji. Torej operacija `POVIŠAJ(`u`)` očitno ohranja h kot višinsko funkcijo.

Ostane nam pokazati še, da če je h višinska funkcija pred operacijo `POTISNI(`u`,`v`)`, potem je tudi po operaciji. Med to operacijo se zgodi natanko ena izmed naslednjih stvari:

(1) *Dodamo residualno povezavo (v, u) v E_f.* V tem primeru imamo $h(v) = h(u) - 1 < h(u) + 1$, torej h ostane višinska funkcija.
(2) *Odstranimo residualno povezavo (u, v) iz E_f.* Z odstranitvijo residualne povezave (u, v) pravzaprav izgubimo zahtevo iz definicije višinske funkcije (definicija 2.3), tako da h na prazno ostane višinska funkcija.

Ker vse operacije ohranjajo lastnosti višinske funkcije funkcije h, po principu indukcije sledi lema. □

Sledi lema, ki nam da pomembno posledico definicije višinske funkcije.

Lema 3.5. *Naj bo $G = (V, E, s, t)$ pretočno omrežje, f predtok v G in h višinska funkcija na V. Potem ne obstaja pot od s do t v residualnem omrežju G_f.*

Dokaz. Pokažimo s pomočjo protislovja. Recimo torej, da v G_f obstaja pot p od s do t, kjer je $p = \langle v_0 = s, v_1, v_2, \ldots v_{n-1}, v_n = t\rangle$. Brez škode za splošnost lahko predpostavimo, da je p enostavna pot, torej pot brez ciklov. Potem velja $n < |V|$. Ker je p pot v residualnem omrežju G_f, za vsak $i = 0, 1, \ldots, n-1$ velja, da je $(v_i, v_{i+1}) \in E_f$. Ker pa je h višinska funkcija, za vsak $i = 0, 1, \ldots, n-1$ velja $h(v_i) \leq h(v_{i+1}) + 1$. Od tod sledi, da velja $h(s) \leq h(t) + n$. Ampak, ker je po definiciji višinske funkcije $h(t) = 0$, dobimo $h(s) \leq n < |V|$, kar pa je v protislovju s tem, da je h višinska funkcija. Veljati bi namreč moralo $h(s) = |V|$. Zaključimo, da v residualnem omrežju G_f torej ne obstaja pot med s in t. □

Sedaj smo pridobili vso potrebno znanje, da dokažemo, da ČE se algoritem `POTISNI-POVIŠAJ` zaključi, je potem predtok, ki ga algoritem vrne, enak maksimalnemu pretoku skozi omrežje.

Izrek 3.6. *Naj bo $G = (V, E, s, t)$ pretočno omrežje. Če poženemo algoritem `POTISNI-POVIŠAJ` na pretočnem omrežju G in se ustavi, potem je predtok f, ki ga algoritem vrne, enak maksimalnemu toku skozi pretočno omrežje G.*

Dokaz. Najprej pokažimo, da je predtok f na vsaki iteraciji zanke `DOKLER` v vrstici 2 algoritma `POTISNI-POVIŠAJ` res predtok. Očitno je pred prvo iteracijo f predtok, saj za to poskrbi operacija `INICIALIZIRAJ_PREDTOK`. Znotraj zanke `DOKLER` se lahko zgodita le dve operaciji, ali `POTISNI` ali `POVIŠAJ`. Operacija `POVIŠAJ` vpliva le ne višine vozlišč in ne na vrednosti $f(u, v)$. Posledično f ostane predtok.

Med operacijo `POTISNI` pride do sprememb vrednosti $f(u, v)$. Poskrbeti moramo, da se ohranita lastnosti predtoka:

(1) Za vsako povezavo $(u, v) \in E$ velja $f(u, v) \leq c(u, v)$.
(2) Za vsako vozlišče $u \in V$ velja $e(u) = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) \geq 0$.

Ker smo v vrstici 3 operacije `POTISNI` vzeli $\Delta = \min\{e(u), c(u, v) - f(u, v)\}$, vidimo, da bomo ne glede na to, kaj bo minimum, uspeli zadostiti pogoju (1). Namreč, če vzamemo $c(u, v) - f(u, v)$, bomo povezavo (u, v) ravno nasičili in bo po opravljeni operaciji veljalo $f(u, v) = c(u, v)$. Če pa velja $e(u) < c(u, v) - f(u, v)$, pa povezave ne bomo zasičili in tako niti ne bomo presegli njene kapacitete.

Podobno premislimo, da se ohrani lastnost (2). Če v vrstici 3 operacije `POTISNI` vzamemo $\Delta = e(u)$, bomo potem v vrstici 7 zmanjšali presežek v vozlišču u na nič, torej bo veljalo $e(u) = 0$. Če pa bomo vzeli $\Delta = c(u, v) - f(u, v) < e(u)$, bo po opravljeni operaciji veljalo $\tilde{e}(u) = e(u) - \big(c(u, v) - f(u, v)\big) > 0$, kjer je $\tilde{e}(u)$ presežek po opravljeni operaciji. Ker $e(v)$ prištejemo pozitivno vrednost, bo po opravljeni operaciji še vedno veljalo $e(v) > 0$.

Ob zaključku izvajanja algoritma velja $e(u) = 0$ za vsak $u \in V \setminus \{s, t\}$. Namreč iz leme 3.2 in dejstva, da je f po vsaki operaciji še vedno predtok, sledi, da po zaključku ne morejo obstajati vozlišča s presežki (s in t namreč nikoli nista v presežku). To pomeni, da je predtok f pravzaprav tok. Lema 3.4 nam pove, da je ob zaključku algoritma h še vedno višinska funkcija, od koder po lemi 3.5 sledi, da ob

zaključku algoritma ni poti med s in t v residualnem omrežju G_f. Po izreku (REFERENCA NA MAX-FLOW MIN-CUT, KI GA JE POTREBNO VKLJUČITI) (izrek o maksimalnem pretoku in minimalnem prerezu) je tako f maksimalni pretok. □

S tem smo pokazali, da če se algoritem konča, dobimo pravilen rezultat. Ostane nam pokazati še, da se algoritem sploh konča. To bomo naredili v naslednjem podrazdelku, v katerem se bomo ukvarjali s časovno zahtevnostjo. Omejili bomo število operacij, ki se lahko zgodijo, in s tem pokazali, da se algoritem res konča.

3.5. **Časovna zahtevnost algoritma.** V tem podrazdelku bomo kot obljubljeno dokazali še, da se algoritem `POTISNI-POVIŠAJ` konča. To bomo naredili s pomočjo omejevanja števila operacij, ki se lahko zgodijo. Namesto osnovnih dveh operacij, `POTISNI` in `POVIŠAJ`, se bomo tukaj ukvarjali s tremi operacijami – operacijo `POTISNI` bomo namreč razdelili na dve, na tisto, ki povezavo zasiči, in tisto, ki je ne.

Predno se lotimo analize časovne zahtevnosti pa si poglejmo in dokažimo še eno lemo. Spomnimo se, da namreč dovolimo povezave v izvir s v residualnem omrežju.

Lema 3.7. *Naj bo $G = (V, E, s, t)$ pretočno omrežje in f predtok v G. Potem za vsako vozlišče $x \in V$, ki je v presežku, obstaja enostavna pot od x do s v residualnem omrežju G_f.*

Dokaz. Za vozlišče v presežku x, definirajmo

$$U = \{v : \text{obstaja enostavna pot od } x \text{ do } v \text{ v } G_f\}.$$

Predpostavimo, da $s \notin U$, in pokažimo protislovje. Definirajmo še $\overline{U} = V \setminus U$. □

Slovar strokovnih izrazov

Literatura