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

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. Graf G je par množic $G = (V, E)$, kjer je G množica vozlišč grafa, E pa je množica povezav grafa G.

Definicija 2.2. Naj bo $G = (V, E)$ graf. **Omrežje** na grafu G je par (G, c), kjer je $c\colon V \times V \to \mathbb{R}_+ \cup \{\infty\}$ **funkcija prepustnosti**, ki vsaki povezavi (u, v) priredi njeno prepustnost $c(u, v)$. Prepustnost $c(u, v) = \infty$ natanko tedaj, ko prepustnost povezave ni omejena.

Rekli bomo še, da $c(u, v) = 0$ natanko tedaj, ko povezava ne obstaja.

Definicija 2.3. Naj bo $G = (V, E)$ graf in (G, c) omrežje na grafu G. **Pretočno omrežje** na omrežju (G, c) je četverica (G, c, s, t), kjer je $s \in V$ začetno vozlišče pretočnega omrežja, rečemo mu **izvir**, $t \in V$ pa končno vozlišče pretočnega omrežja, ki mu pravimo **ponor**.

Definicija 2.4. Psevdopretok je funkcija $f\colon V \times V \to \mathbb{R}$, ki zadošča pogojema

(1) Za vsaki vozlišči $u, v \in V$ velja $f(u, v) = -f(v, u)$.
(2) Za vsaki vozlišči $u, v \in V$ velja $f(u, v) \leq c(u, v)$, kjer je c funkcija prepustnosti.

Definicija 2.5. Residualna prepustnost povezave glede na trenuten psevdopretok f je funkcija $c_f\colon V \times V \to \mathbb{R}_+$, definirana kot razlika prepustnosti povezave in trenutnega toka preko nje. Velja torej $c_f(u, v) = c(u, v) - f(u, v)$.

Definicija 2.6. Funkcija presežka za psevdopretok f je funkcija $e_f\colon V \to \mathbb{R}$, definirana z $e_f(u) = \sum_{v \in V} f(v, u)$. Če je $e_f(u) > 0$, pravimo, da je u **v presežku**.

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

Definicija 2.7. Predpretok f je tak psevdopretok, v katerem za vsak $v \in V \setminus \{S\}$ velja, da je neto tok, ki priteče v vozlišče v, nenegativen, torej da velja $e_f(v) \geq 0$.

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

Definicija 2.8. Pretok f je tak psevdopretok, v katerem za vsak $v \in V \setminus \{s, t\}$ velja, da je neto tok, ki priteče v vozlišče v, enak nič, torej da velja $e_f(v) = 0$.

Definicija 2.9. Vrednost pretoka f je tok, ki vstopa v ponor t. Označimo ga z $|f|$. Velja torej $|f| = e_f(t)$.

Definicija 2.10. Maksimalni pretok je pretok f, za katerega velja

$$|f| = \max_{f_i} |f_i|.$$

Naslednja definicija nam bo dala nov atribut vozlišč.

Definicija 2.11. 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. Sledila bo implementacija algoritma z rahlo izboljšavo v programskem jeziku C++. Pokazali bomo pravilnost delovanja algoritma in njene implementacije ter časovno zahtevnost algoritma in implementacije.

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 *predpretok*. 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 predpretoka*. Zapišimo psevdokodo za to operacijo.

```
INICIALIZIRAJ_PREDPRETOK(G,s)
 1   // V grafu G si izberemo vozlišče s
 2   // in inicializiramo predpretok.
 3   ZA vsak v v V(G)
 4       h(v) = 0
 5       e(v) = 0
```

```
 6   ZA vsak (u,v) v E(G)
 7       f(u,v) = 0
 8   h(s) = |V|
 9   ZA vsak v, za katerega obstaja (s,v) v E(G)
10       f(s,v) = c(s,v)
11       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 predpretoka 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_PREDPRETOK(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

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

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 C++.** Najprej si poglejmo idejo implementacije. Prvo vprasanje, ki se nam postavi, je, kako bomo predstavili graf. Za vsako vozlišče si moramo zapomniti njegovo višino in presežek toka, ki se v njem nahaja. Odločimo se, da bomo vozlišče predstavili s strukturo, saj bo tako koda bolj berljiva. Lahko bi sicer uporabili razred, vendar ni potrebe, lahko pa bi naredili tudi dva vektorja, enega, ki bi predstavljal višinsko funkcijo in drugega, ki bi povedal, koliko presežka toka je v vsakem izmed vozlišč. Za predstavitev grafa bomo tako uporabili vektor vozlišč in pa matriko $p \in \{0,1\}^{V\times V}$ booleanovih vrednosti. Element p_{ij} bo povedal, ali je povezava ij v grafu ali ne. Prav tako si bomo definirali dve matriki $c, f \in \mathbb{N}_0^{V\times V}$. Matrika c bo imela na ij-tem mestu kapaciteto povezave ij, matrika f pa pretok preko povezave ij.

Definirali si bomo še vrsto, v katero bomo spravljali vozlišča, ki imajo presežek toka. S tem bomo lahko do elementov, na katerih moramo opraviti še operacijo potiska ali povišanja dostopali v konstantnem času, algoritem pa bo deloval pravilno. Vzeli bomo namreč prvo vozlišče v vrsti, na njem naredili ali `POTISNI` ali `POVISAJ`, in ga odstranili iz vrste, če bo po opravljeni operaciji njegov presežek ničeln. Prav tako bomo na konec vrste dodali vozlišče, v katerega smo potisnili tok, če je seveda šlo za operacijo potiska. Na tej točki bomo potrebovali še vektor booleanovih vrednosti, ki bo za vsako vozlišče povedal, ali smo ga že dodali med presežke. S tem se izognemo podvajanju elementov v vrsti presežkov. S tem pripomoremo k hitrejšemu delovanju algoritma; in hkrati tudi pravilnejšemu, saj se lahko zgodi, da bi algoritem želel opravljati potisk ali povišanje na vozlišču, ki smo ga spraznili, ko se je prejšnjič pojavil v vrsti.

Algoritem bomo razbili na več funkcij. Glavne bodo seveda `potisni_povisaj`, `potisni` in `povisaj`. Poglejmo si opis, kaj počne katera izmed funkcij.

- `potisni_povisaj()`: Izračuna maksimalni pretok v grafu. Najprej inicializira predpretok, nato pa izvaja zanko, dokler obstaja vozlišče s presežkom. Vsakič se odloči, ali bo naredila potisk ali povišanje in potisk. Za povišanjem naredimo takoj potisk zato, da zmanjšamo število vrtljajev zanke in nam tako ni potrebno še enkrat računati najnižjega soseda, saj bo ostal isti.
- `inicializiraj_predpretok()`: Inicializira predpretok. Nastavi višino vozlišča s na moč množice vozlišč. Nato potisne tok po vseh sosednjih povezavah in posodobi presežke v vozliščih. Vmes tudi doda residualno (obratno) povezavo, katere kapaciteta je 0, tok po njej pa je enak negativno predznačenemu toku, ki smo ga poslali po originalni povezavi. Če vozlišče, v katerega smo potisnili tok, ni t, ga dodamo med vozlišče s presežki in označimo, da je bil tja dodan.
- `potisni(u,v)`: Potisne tok iz vozlišča u v vozlišče v. Tok, ki ga lahko potisne, je enak minimumu med presežkom v vozlišču u in residualno kapaciteto povezave (u, v). Če smo vozlišče u izpraznili, ga odstranimo iz vrste presežkov in označimo, da ga ni več notri. Če vozlišče v ni t ali s in še ni bilo dodano med presežke, ga dodamo.
- `povisaj(u,h)`: Poveča višino vozlišča u za $h + 1$.
- `najnizji_sosed(u)`: Poišče tisto vozlišče v med sosedi u, ki ima najmanjšo višino in povezava (u, v) še ni zasičena.
- `napolni_graf`: Inicializira vse potrebne spremenljivke. V vektor vozlišč doda vozlišča z višino in presežkom nič. V matriko kapacitet vpiše pripadajoče kapacitete ter v matriki p označi, katere povezave obstajajo.

Oglejmo si sedaj implementacijo.

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

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

#include<iostream>
#include<queue>
#include<cstdio>
#include<cstdlib>
#include<climits>

using namespace std;

// Vozlisce predstavimo s strukturo. Za vsako vozlisce si zapomnimo
// njegovo visino in presezek toka v njem.
struct Vozlisce
{
    int h, e;

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

// Prototipi funkcij.
int potisni_povisaj();
void inicializiraj_predpretok();
void potisni(int u, int v);
void povisaj(int u, int h);
int najnizji_sosed(int u);
void napolni_graf();

// Globalne spremenljivke.

// Graf predstavimo z vektorjem vozlisc.
vector<Vozlisce> vozlisca;

// V vrsto dajemo vozlisca, ki so v presezku. S tem lahko v
// O(1) dostopamo do naslednjega vozlisca, na katerem je potrebno
// opraviti operacijo POTISNI ali POVISAJ.
queue<int> presezki;
```

```
// V vektor shranjujemo vozlisca, ki smo jih ze dodali med
// presezke. S tem dosezemo, da se vsako vozlisce v presezkih
// pojavi kvecjemu enkrat.
vector<bool>viden;

// Matrika povezav grafa. Ce je p[i][j] true, to pomeni, da
// povezava obstaja. S tem se izognemo temu, da bi se v sosedih
// vozlisc zaceli sosedi ponavljati. Prej smo namrec za vsako
// vozlisce imeli vektor sosedov in s tem ponazorili povezave.
// V obeh primerih s tem zmanjsamo casovno zahtevnost iz O(E)
// na O(V).
bool** p;

int** c; // Matrika kapacitet povezav.
int** f; // Matrika toka.

//=====================================================================
//
//        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();

    // Da preverimo, ali se oddani tok ujema s prejetim, da ga ni
    // slucajno ostalo kaj vmes. (Za nekaksno lahko preverjanje.)
    cout << "e(s) = " << vozlisca[0].e << endl;

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

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

// Glavna funkcija algoritma potisni-povisaj. Najprej
// inicializira predpretok, 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_predpretok();

    // Dokler ima katero izmed vozlisc presezek toka,
    // moramo opraviti ali POTISNI ali POVISAJ. Ce opravimo
    // POVISAJ, lahko takoj potem tudi potisnemo na tisto
    // vozlisce. S tem si zmanjsamo stevilo zank.
    while (presezki.size() > 0)
    {
        int u = presezki.front();
        int v = najnizji_sosed(u);

        if (vozlisca[u].h == 1 + vozlisca[v].h)
            potisni(u, v);
        else
        {
            povisaj(u, vozlisca[v].h);
            potisni(u,v);
        }
    }

    // Vrnemo presezek v vozliscu t. Lahko bi vrnili
    // tudi -presezek v vozliscu s.
    return vozlisca.back().e;
}

// Inicializira predpretok. Visino vozlisca s nastavi na
// |V|, zasici povezave iz s, doda residualne povezave
// in ce sosed ni vozlisce t, ga doda v presezke.
void inicializiraj_predpretok()
{
    vozlisca[0].h = vozlisca.size();

    for (int v = 0; v < vozlisca.size(); v++)
    {
        if (p[0][v])
        {
            // Zasici povezavo.
            f[0][v] = c[0][v];

            // Nastavi presezek v sosedu.
            vozlisca[v].e = f[0][v];

            // Doda residualno povezavo.
            f[v][0] -= f[0][v];
            p[v][0] = true;

            // V vozliscu s posodobi oddani tok.
            vozlisca[0].e -= f[0][v];
```

```
            // Ce vozlisce ni t, ga doda v presezke.
            if (v != vozlisca.size()-1)
            {
                presezki.push(v);
                viden[v] = true;
            }
        }
    }
}

// Vrne indeks najnizjega soseda.
int najnizji_sosed(int u)
{
    int sosed;
    int min_visina = INT_MAX;

    for (int v = 0; v < vozlisca.size(); v++)
    {
        if (p[u][v])
        {
            // Ce je visina soseda manjsa od trenutne najmanjse
            // visine in ce povezava ni zasicena, potem je to
            // kandidat za najnizjega soseda.
            if (vozlisca[v].h < min_visina && c[u][v] - f[u][v] > 0)
            {
                min_visina = vozlisca[v].h;
                sosed = v;
            }
        }
    }

    return sosed;
}

// Operacija POTISNI. Potisne lahko minimum med presezkom
// v vozliscu in residualno kapaciteto povezave.
void potisni(int u, int v)
{
    int delta = min(vozlisca[u].e, c[u][v] - f[u][v]);

    // Posodobi presezek v krajiscih povezave.
    vozlisca[u].e -= delta;
    vozlisca[v].e += delta;

    // Posodobi tok prek povezave in residualne povezave.
    f[u][v] += delta;
    f[v][u] -= delta;
```

```
    // Doda obratno povezavo.
    p[v][u] = true;

    // Ce smo z vozliscem opravili, ga odstranimo iz presezkov.
    if (vozlisca[u].e == 0)
    {
        presezki.pop();
        viden[u] = false;
    }

    // Ce konec povezave ni t ali s in ce vozlisce se ni v
    // presezkih, ga doda v presezke.
    if (v != vozlisca.size()-1 && v != 0 && !viden[v])
    {
        presezki.push(v);
        viden[v] = true;
    }
}

// Operacija POVISAJ. Vozliscu nastavi visino na prej
// izracunano minimalno visino "dobrih" sosedov h + 1.
void povisaj(int u, int h)
{
    vozlisca[u].h = h + 1;
}

// Iz datoteke prebere podatke o stevilu vozlisc in
// povezavah. V vektor vozlisc doda vozlisca, ki jim
// nastavi visino in presezek na 0. Vsakemu vozliscu
// nato doda sosede in nastavi kapaciteto povezav.
void napolni_graf()
{
    int V;
    scanf("%d\n", &V);

    c = (int**) malloc(V*sizeof(int*));
    f = (int**) malloc(V*sizeof(int*));
    p = (bool**) malloc(V*sizeof(int*));

    for (int i = 0; i < V; i++)
    {
        vozlisca.push_back(Vozlisce(0, 0));
        viden.push_back(false);
        c[i] = (int*) malloc(V*sizeof(int));
        f[i] = (int*) malloc(V*sizeof(int));
        p[i] = (bool*) malloc(V*sizeof(bool));
    }

    int u, v, kapaciteta;
```

```
    while (scanf("%d %d %d\n", &u, &v, &kapaciteta) != EOF)
    {
        c[u][v] += kapaciteta;
        p[u][v] = 1;
    }
}
```

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).

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, $f\colon V \times V \to \mathbb{N}_0$ predpretok 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 predpretok, 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 2.11.*

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

Po inicializaciji predpretoka 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 2.11), 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 *predpretok 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 predpretok, 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 predpretok f, ki ga algoritem vrne, enak maksimalnemu toku skozi pretočno omrežje G.*

Dokaz. Najprej pokažimo, da je predpretok f na vsaki iteraciji zanke `DOKLER` v vrstici 2 algoritma `POTISNI-POVIŠAJ` res predpretok. Očitno je pred prvo iteracijo f predpretok, saj za to poskrbi operacija `INICIALIZIRAJ_predpretok`. 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 predpretok.

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

(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 predpretok, sledi, da po zaključku ne morejo obstajati vozlišča s presežki (s in t namreč nikoli nista v presežku). To pomeni, da je predpretok 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 predpretok 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 $\overline{U} = V \setminus U$. Velja torej $V = U \cup \overline{U}$. Spomnimo se še, da je $e(u) = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v)$. Tako velja

$$\begin{aligned}
\sum_{u \in U} e(u) &= \sum_{u \in U} \left(\sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) \right) \\
&= \sum_{u \in U} \left(\left(\sum_{v \in U} f(v, u) + \sum_{v \in \overline{U}} f(v, u) \right) - \left(\sum_{v \in U} f(u, v) + \sum_{v \in \overline{U}} f(u, v) \right) \right) \\
&= \sum_{u \in U} \sum_{v \in U} f(v, u) + \sum_{u \in U} \sum_{v \in \overline{U}} f(v, u) - \sum_{u \in U} \sum_{v \in U} f(u, v) - \sum_{u \in U} \sum_{v \in \overline{U}} f(u, v) \\
&= \sum_{u \in U} \sum_{v \in \overline{U}} f(v, u) - \sum_{u \in U} \sum_{v \in \overline{U}} f(u, v).
\end{aligned}$$

Zadnja enakost sledi iz dejstva, da seštevamo po povezavah, ki imajo krajišča v isti množici. Torej dobimo v vsoti tako $f(u, v)$ kot tudi $f(v, u)$. Ker velja $f(u, v) = -f(v, u)$, velja

$$\sum_{u \in U} \sum_{v \in U} f(u, v) = \sum_{u \in U} \sum_{v \in U} f(v, u) = 0.$$

Vemo, da je $\sum_{u \in U} e(u) > 0$, saj $x \in U$ in $e(x) > 0$ ter $e(u) \geq 0$ za vse $y \in V$, razen s, za katerega smo predpostavili $s \notin U$. Torej

$$\sum_{u \in U} \sum_{v \in \overline{U}} f(v, u) - \sum_{u \in U} \sum_{v \in \overline{U}} f(u, v) > 0.$$

Ker je tok po vseh povezavah nenegativen, mora veljati $\sum_{u \in U} \sum_{v \in \overline{U}} f(v, u) > 0$. To pomeni, da obstaja vozlišči $u' \in U$ in $v' \in \overline{U}$, za kateri velja $f(v', u') > 0$. Ampak to pomeni, da obstaja residualna povezava (u', v'). To pa pomeni, da obstaja enostavna pot od x do v', namreč $x \rightsquigarrow u' \to v'$. Pot $x \rightsquigarrow u'$ namreč obstaja po definiciji U. To pa je v protislovju z definicijo U. □

Z naslednjo lemo bomo omejili največjo možno višino vozlišč, njena posledica pa bo omejila skupno število opravljanj operacije `POVISAJ`.

Lema 3.8. *Naj bo $G = (V, E, s, t)$ pretočno omrežje. Na koncu izvajanja algoritma `POTISNI-POVISAJ` na G, za vsak $u \in V$ velja $h(u) \leq 2|V| - 1$.*

Opomba 3.9. Vemo, da višina vozlišč med izvajanjem algoritma `POTISNI-POVISAJ` ne pada. To pomeni, da je v vsakem trenutku izvajanja algoritma $h(u) \leq 2|V| - 1$ za vsak $u \in V$.

Dokaz. Višini vozlišč s in t se ne spreminjata. Velja $h(s) = |V| \leq 2|V| - 1$ in $h(t) = 0 \leq 2|V| - 1$.

Osredotočimo se torej na vozlišča $u \in V \setminus \{s, t\}$. Na začetku velja $h(u) = 0 \leq 2|V| - 1$. Pokažimo, da po vsaki operaciji `POVISAJ` še vedno velja $h(u) \leq 2|V| - 1$. Vsakič, ko povišamo vozlišče u, ima u presežek toka. Lema 3.7 nam pove, da obstaja enostavna pot p od u do s v residualnem grafu G_f. Naj bo $p = \langle v_0 = u, v_1, \ldots, v_{n-1}, v_n = s \rangle$. Velja $n \leq |V| - 1$, saj je p enostavna pot. Velja tudi, da je za vsak $i = 0, 1, \ldots, n-1$ povezava $(v_i, v_{i+1}) \in E_f$. Po lemi 3.4 velja $h(v_i) = h(v_{i+1}) + 1$. Če združimo te enakosti po celi poti p, dobimo

$$h(u) = h(v_0) \leq h(v_n) + n = h(s) + n \leq h(s) + |V| - 1 = 2|V| - 1.$$

□

Posledica 3.10 (omejenost števila operacij `POVISAJ`)**.** *Naj bo $G = (V, E, s, t)$ pretočno omrežje. Potem je število operacij* ***POVISAJ*** *med izvajanjem algoritma* ***POTISNI_POVISAJ*** *manjše od $2|V|^2$.*

Dokaz. Povišamo lahko samo $|V| - 2$ vozlišč; vozlišči s in t imata namreč ves čas isto višino. Za vozlišče $u \in V \setminus \{s, t\}$ na začetku velja $h(u) = 0$. Lema 3.8 nam pove, da vozlišče u ne preseže višine $2|V| - 1$ in ker višina ne pada, ga lahko povišamo največ $(2|V| - 1)$-krat. Ker se to lahko zgodi za $|V| - 2$ vozlišč, je število operacij `POVISAJ` tako navzgor omejeno z

$$(|V| - 2)(2|V| - 1) = 2|V|^2 - 5|V| + 2 < 2|V|^2.$$

□

Ostane nam pokazati še, da je tudi število operacij `POTISNI` omejeno. Kot že rečeno, bomo dokaz razbili na dva dela. Najprej bomo pokazali, da je omejeno število potiskov, ki povezavo zasičijo, potem pa bomo pokazali še, da je omejeno tudi število potiskov, ki povezave ne zasičijo.

Lema 3.11. *Naj bo $G = (V, E, s, t)$ pretočno omrežje. Potem je število operacij* ***POTISNI****, ki ne zasičijo povezave, med izvajanjem algoritma* ***POTISNI_POVISAJ*** *manjše od $2|V||E|$.*

Dokaz. Za vsak par vozlišč $u, v \in V$ bomo prešteli, koliko potiskov, ki nasičijo povezavo, je iz u v v in obratno. Če kakšen tak potisk obstaja, je vsaj ena od povezav (u, v) in (v, u) v E.

Recimo, da se je zgodil potisk iz u v v. Na tej točki je veljalo $h(v) = h(u) - 1$. Če želimo še kdaj potisniti iz u v v, moramo najprej potisniti iz v v u nekaj toka, saj je povezava sedaj zasičena, in veljati mora $u(v) = u(h) + 1$. Ker $h(u)$ ne pada, se mora $u(v)$ povečati za vsaj 2. Podobno premislimo, da se mora $h(u)$ povečati vsaj za 2 med potiski iz v v u, ki zasičijo povezavo. Spet uporabimo lemo 3.8, ki nam pove, da so višine vozlišč ves čas med 0 in $2|V| - 1$. Iz tega sledi, da se lahko $h(u)$ poveča za 2 manj kot $|V|$-krat med delovanjem algoritma. Ker se mora med dvema potiskoma, ki zasičita povezavo, vsaj enkrat eden od $h(u)$ in $h(v)$ povečati za 2, je tako možnih največ $2|V|$ potiskov med u in v, ki zasičijo povezavo (lahko jih je tudi manj, če se $h(u)$ oziroma $h(v)$ povečata večkrat). Ker je povezav $|E|$, tako dobimo, da je potiskov, ki zasičijo povezavo, največ $2|V||E|$. □

Lema 3.12 (omejenost števila operacij POTISNI, ki ne zasičijo povezave)**.** *Naj bo $G = (V, E, s, t)$ pretočno omrežje. Potem je število operacij* `POTISNI`*, ki ne zasičijo povezave, med izvajanjem algoritma* `POTISNI_POVISAJ` *manjše od $4|V|^2(|V| + |E|)$.*

Že pogled na zgornjo mejo nas prepriča, da bo dokaz zgornje leme težji oziroma kompleksnejši, kot dokaz prejšnjih. Pa se ga lotimo.

Dokaz. Definirajmo najprej potencial Φ kot

$$\Phi = \sum_{v:e(v)>0} h(v),$$

torej kot vsoto višin vozlišč v presežku. Na začetku je $\Phi = 0$. Opazimo, da se vrednost Φ lahko spremeni po povišanju, potisku, ki zasiči povezavo, in potisku, ki povezave ne zasiči. Najprej bomo omejili, za koliko se lahko Φ poviša po potisku, ki zasiči povezavo, in povišanju. Potem bomo pokazali, da se po vsakem potisku, ki povezave ne zasiči, vrednost potenciala Φ zmanjša vsaj za 1. To znanje bomo potem uporabili za izračun zgornje meje števila potiskov, ki povezave ne zasičijo.

Oglejmo si najprej oba načina, na katera se potencial Φ lahko poveča. Povišanje vozlišča očitno lahko poveča Φ za največ $2|V| - 1 < 2|V|$, saj po tej operaciji ostane množica, po kateri seštevamo, enaka, vozlišče pa ima po lemi 3.8 lahko največjo višino $2|V| - 1$, za kolikor se mu lahko tudi največ poviša. Potisk, ki zasiči povezavo (u, v) lahko poveča Φ za največ $2|V| - 2 < 2|V|$. Po tej operaciji se namreč množica, po kateri seštevamo, kvečjemu poveča, saj vanjo vstopi v, u pa v njej lahko ostane ali pa ne. Če ostane, se tako lahko Φ poveča za višino vozlišča v, ki pa je lahko največ $2|V| - 2$ (največja možna višina vozlišč je po lemi 3.8 enaka $2|V| - 1$, vendar je vsaj vozlišče u višje od v, drugače potisk ne bi bil mogoč). Če pa u ne ostane v množici, torej ni več v presežku (zgodilo se je, da je veljalo $e(u) = c(u, v) - f(u, v)$), pa se Φ celo pomanjša, saj smo iz množice, po kateri seštevamo, odstranili vozlišče z večjo višino kot je višina vozlišča, ki smo ga v množico dodali.

Preostane nam obravnavati potisk, ki povezave ne zasiči. Pred potiskom je bil u v presežku, vozlišče v pa je lahko bilo v presežku, lahko pa tudi ne. Po operaciji u ne bo več v presežku, saj pri potisku, ki povezave ne zasiči, velja $\Delta = e(u)$, torej je po potisku $e(u) = 0$. Vozlišče v pa je po operaciji v presežku, razen če $v = s$. To pomeni, da se je Φ zaradi operacije zmanjšal za $h(u)$, povečal pa za ali 0 ali $h(v)$. Ker velja $h(u) - h(v) = 1$, se je Φ skupno zmanjšal za vsaj 1.

Tako lahko iz zgornjih ugotovitev glede povečanja potenciala Φ, posledice 3.10 in leme 3.11 zaključimo, da se lahko Φ poveča za največ

$$(2|V|)(2|V|^2) + (2|V|)(2|V||E|) = 4|V|^2(|V| + |E|).$$

Ker je $h(u) \geq 0$ za vsak $u \in V \setminus \{s, t\}$ (vozlišči s in t ne moreta biti v presežku), je tudi $\Phi \geq 0$. To pomeni, da se Φ lahko zmanjša za največ $4|V|^2(|V| + |E|)$, kar pa ravno pomeni, da je število potiskov, ki povezave ne zasičijo, enako $4|V|^2(|V| + |E|)$. □

Sedaj pa lahko zgornje ugotovitve združimo v zaključni izrek.

Izrek 3.13 (časovna zahtevnost)**.** *Algoritem* `POTISNI_POVISAJ` *med izvajanjem naredi $\mathcal{O}(V^2E)$ osnovnih operacij.*

Dokaz. Sledi neposredno iz posledice 3.10 ter lem 3.11 in 3.12. □

Slovar strokovnih izrazov

Literatura

[1] T.H. Cormen, C.E. Leiserson, R.L. Rivest in C. Stein, *Introduction to Algorithms*, MIT Press, Massachusetts, 2009.