Goldberg-Tarjanov algoritam za pronalaženje maksimalnog protoka mreže

Matković, Dominik

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

GOLDBERG-TARJANOV ALGORITAM ZA PRONALAŽENJE MAKSIMALNOG PROTOKA MREŽE

Završni rad

Dominik Matković

Osijek, 2020. godina.

1. UVOD

Goldberg-Tarjanov algoritam ili *push-relabel* algoritam je algoritam za izračun maksimalnog protoka u protočnoj mreži. Protočna mreža je usmjereni graf gdje svaka veza ima kapacitet protoka [1]. Količina protoka na svakoj vezi ne može biti veća od kapaciteta te veze. *Push-relabel* naziv dolazi od dvije glavne operacije algoritma. Dizajnirali su ga Andrew V. Goldberg i Robert Tarjan. U ovom radu obrađuje se opća verzija Goldberg-Tarjan algoritma i spominjat će se pojmovi iz teorije grafova, problem maksimalnog protoka, algoritmi maksimalnog protoka i njihov način rada te programski jezik C#. Na kraju rada nalazi se zaključak.

1.1. Zadatak završnog rada

Potrebno je opisati Goldberg-Tarjanov algoritam za pronalaženje maksimalnog protoka, posebno opisati push i relabel operacije, kako se i kada izvode, te što je to preljev toka (*flow excess*). Isti algoritam potrebno je implementirati i isprobati na nekoliko primjera mreže.

2. PROBLEM MAKSIMALNOG PROTOKA

Cilj problema maksimalnog protoka je pronaći najveći mogući protok u protočnoj mreži od izvorne točke do krajnje točke [2]. U ovom problemu vrijede dva pravila:

- Protok veze ne može biti veći od svojeg kapaciteta.
- Suma protoka koji ulaze u točku moraju biti jednaki sumi protoka koje izlaze iz te točke, to vrijedi za sve točke osim za izvornu i krajnju točku.

Tablica 2.1. Algoritmi korišteni za problem maksimalnog protoka

Algoritam	Vremenska složenost
Ford–Fulkerson algoritam [3]	$O(E \max \lvert f \rvert)$
Edmonds–Karp algoritam [3]	$O(VE^2)$
Dinicov algoritam s blokiranjem protoka [4]	$O(V^2E)$
MPM (Malhotra, Pramodh-Kumar i Maheshwari) algoritam [6]	$O(V^3)$
Dinicov algoritam [4]	$O(VElog(V))$
Opći Goldberg-Tarjan algoritam (*push-relabel* algoritam) [5]	$O(V^2E)$
Push-relabel algoritam s *FIFO* odabirom točaka [5]	$O(V^3)$
Push-relabel algoritam s dinamičkim stablima [7]	$O(VElog\frac{V^2}{E})$
KRT (King, Rao, Tarjan) algoritam [8]	$O(EVlog_{\frac{E}{V\log V}}V)$
Binary blocking flow algoritam [1]	$O(E \min\left(V^{\frac{2}{3}}, \sqrt{E}\right) \log\frac{V^2}{E} \log U)$
James B. Orlin + KRT (King, Rao, Tarjan) algoritam [9]	$O(VE)$

3. KAKO RADI ALGORITAM

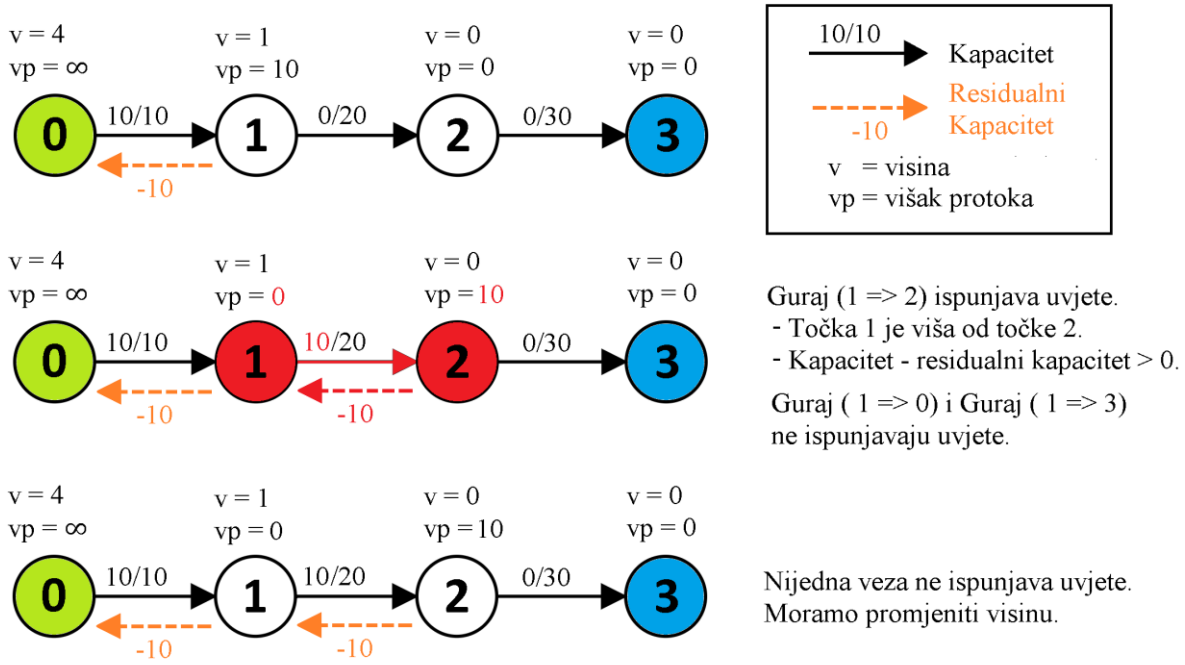
Goldberg-Tarjanov algoritam sastoji se od 4 operacije: guraj *(push)*, promjeni visinu *(relabel)* , inicijalizacija *(preflow)* i preljev toka *(flow excess)*.

U algoritmu se koriste dvije matrice susjedstva (*adjacency matrix*). Matrica kapaciteta, koja nam govori kako su točke povezane i s kojim kapacitetom, i residualna matrica, koja nam govori o kretanju protoka. [1]

3.1. Guraj

Operacija guraj se vrši na točki koja ima višak protoka. Višak protoka se "gura" od te točke do izabrane točke koja ispunjava određene uvjete. Ta točka mora biti viša, mora biti susjedna točka i razlika kapaciteta i residualnog protoka između te dvije točke mora biti veća od nula.

U slučaju da je moguće guranje od jedne točke do više točaka ne postoji poseban način odabira u općem Goldberg-Tarjan algoritmu nego algoritam jednostavno odabire onu točku koja prva dođe na red tijekom izvođenja algoritma. [7]



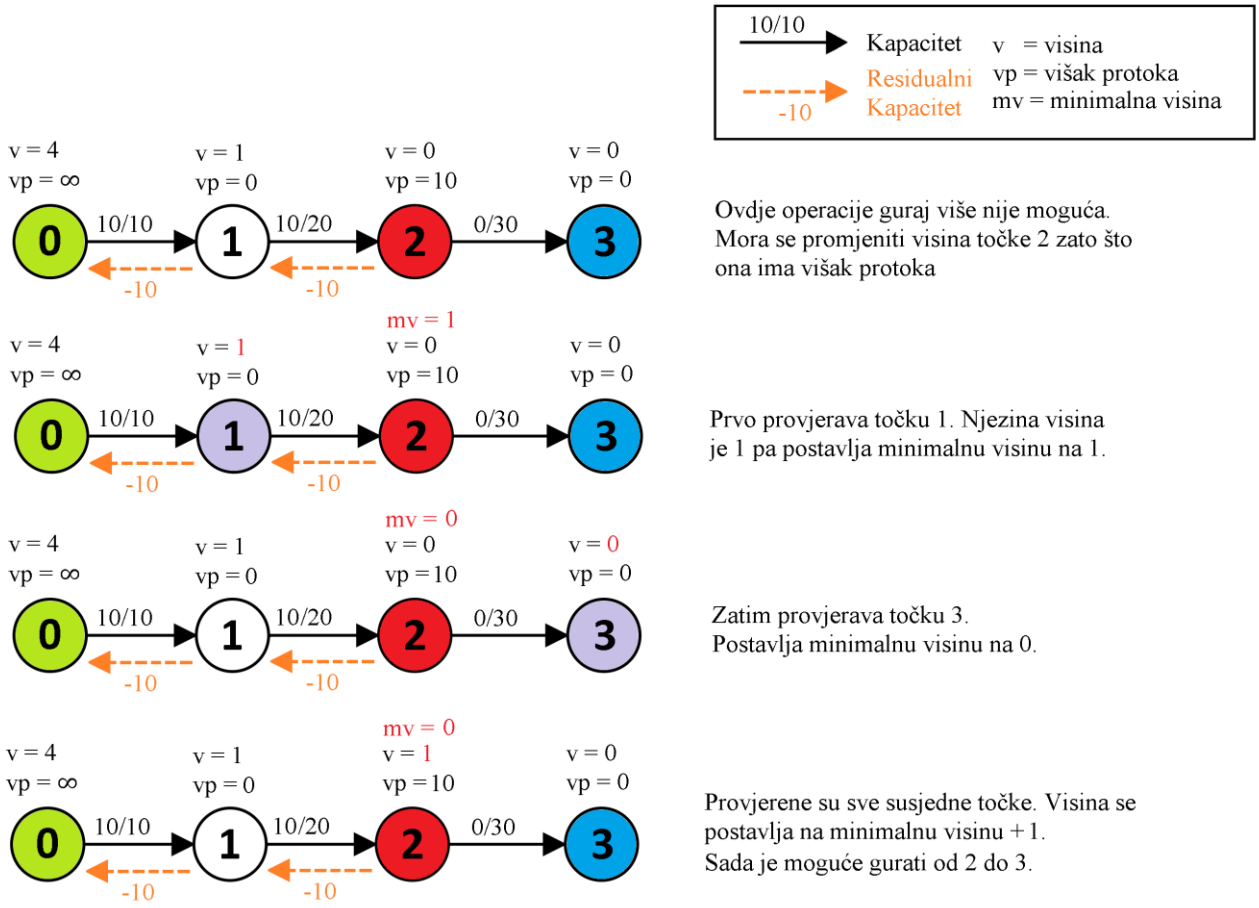
Slika 3.1. Primjer guranja

3.2.Promjeni Visinu

Operacija promjeni visinu se vrši kada operacija guraj nije moguća i vrši se na točkama s viškom protoka.

Operacija provjerava sve susjedne točke i traži najmanju visinu. Kada nađe najmanju visinu točki na kojoj se vrši promjena visine postavlja visinu na najmanju pronađenu visinu plus jedan.

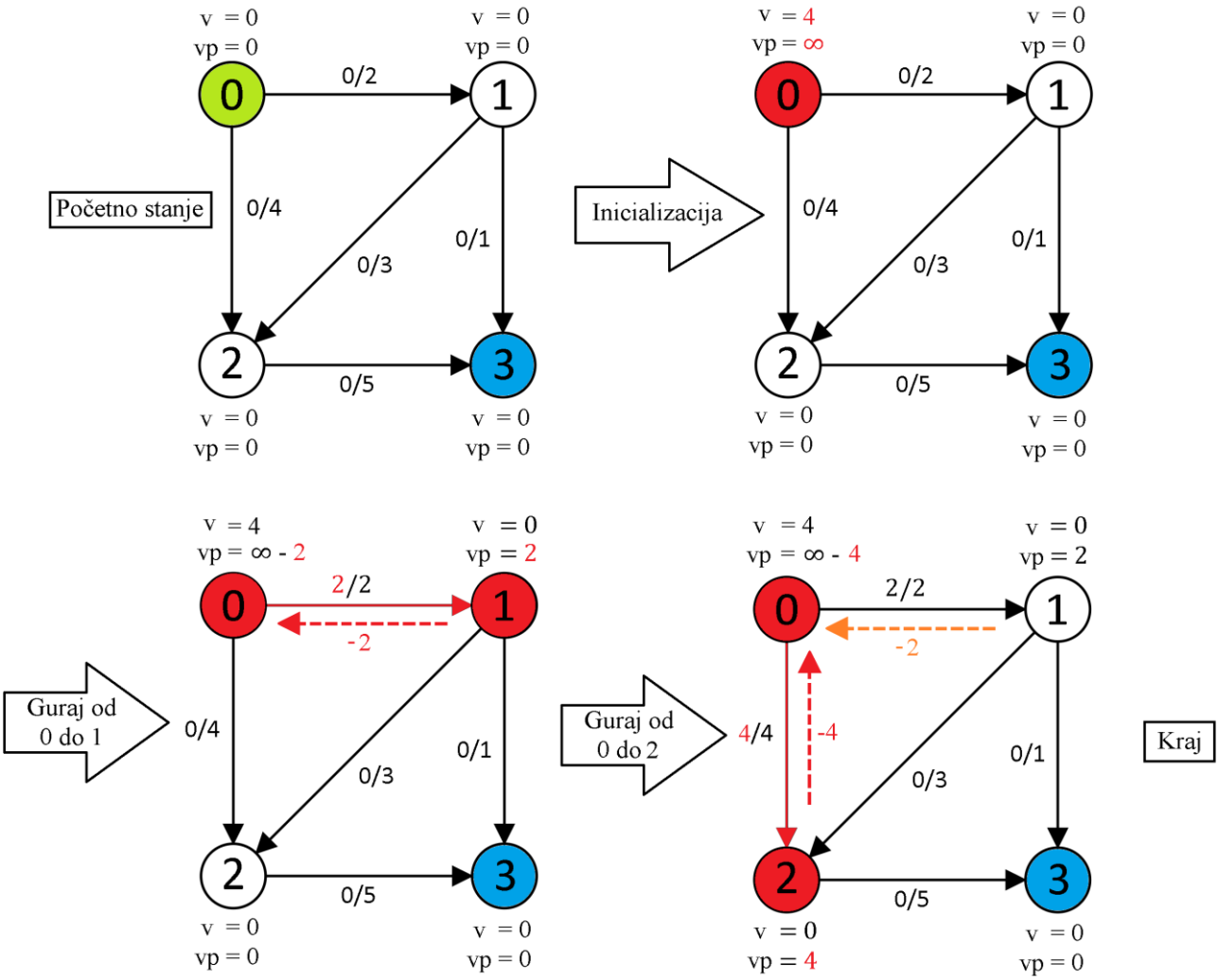
Isto kao i kod guraj operacije ako se algoritam naiđe u situaciji gdje postoje dvije s viškom protoka izabrat će onu koja prva dođe na red. [7]



Slika 3.2. Primjer promjene visine

3.3.Inicijalizacija

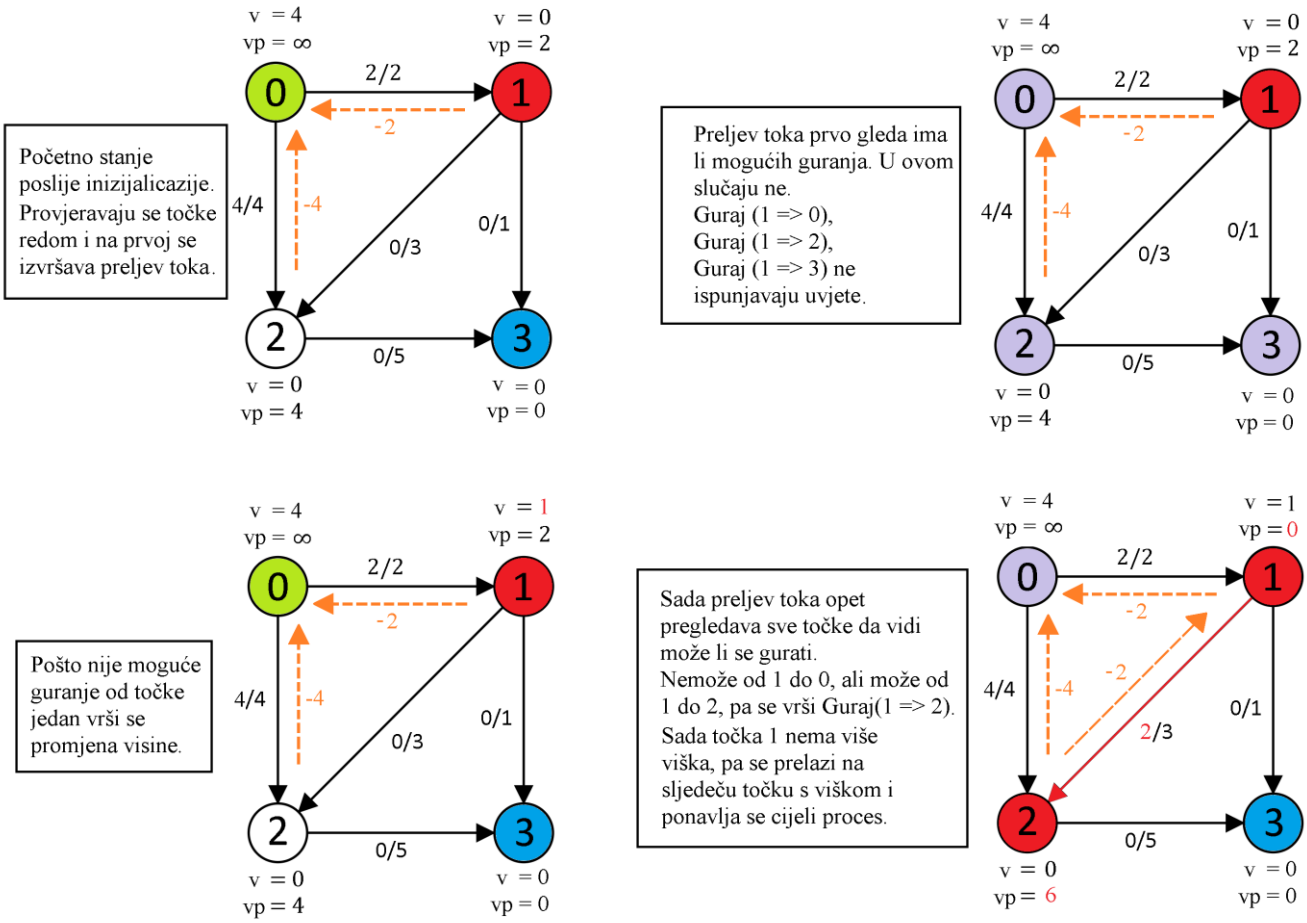
Prvi korak algoritma je inicijalizacija. Inicijalizacija postavlja visine svih točaka na 0 osim izvora koji se postavlja na visinu koja je jednaka broju točaka. Npr. ako ima četiri točke visina izvora će biti 4. Višak protoka izvora se stavlja na najveću moguću vrijednost i onda se vrši operacije guraj na sve susjedne točke izvora.



Slika 3.3. Primjer inicijalizacije

3.4.Preljev Toka

Preljev toka se vrši nad točkama s viškom protoka sve dok nema točaka s viškom protoka osim izvora i kraja. Kada preljeva toka više nije moguć dobivamo maksimalni protok mreže. Preljev toka koristi operacije guraj i promjeni visinu. Provjeravaju se sve točke osim izvora i kraja i na njima se izvodi operacija guraj, ako je moguća. Kada su izvršena sva moguća guranja tada se vrši promjena visine i opet se provjeravanju sve točke sve dok ni guranje i promjena visine nije moguće.



Slika 3.4. Primjer preljeva toka

4. IMPLEMENTACIJA

Algoritam je implementiran u C# programskom jeziku na razvojnom okruženju Visual Studio 2019.

4.1. Deklaracija varijabli

U ovom dijelu koda deklarirane su varijable koje će se koristiti za izvođenje algoritma. Varijabla *n* predstavlja širinu i visinu kvadratne matrice. Varijabla *n* dobiva se korištenjem metode *GetLength* koja vraća broj elemenata u specificiranoj dimenziji. Residualna je matrica koja nam govori o kretanju protoka i istih je dimenzija kao matrica kapaciteta. Niz *visine* sadrži visine svih točaka, niz *visak* sadrži viškove protoka svih točaka. Niz *posjecene* prati koje točke su bile posjećene i koristi se u operaciji preljev toka. Lista *listaTocaka* sadrži sve točke koje nisu izvor i kraj.

```
int n = matricaKapaciteta.GetLength(0);
int[,] residualnaMatrica = new int[n, n];


int[] visine = new int[n];
int[] visak = new int[n];
int[] posjecene = new int[n];


List<int> listaTocaka = new List<int>();
for (int i = 0; i < n; i++)
{
        if (i != kraj && i != izvor)
        {
                listaTocaka.Add(i);
        }
}
```

4.2. Guraj

Operacija guraj uvijek gura maksimalnu moguću količinu protoka. Ta količina određena je varijablom *kolicinaProtoka*, a ona je jednaka cijelom višku protoka na točki ili ako je višak veći od dostupnog kapaciteta veze onda je *kolicinaProtoka* jednaka razlici kapaciteta i residualnog protoka. Kada je određena *kolicinaProtoka* tada se na odredištu dodaje residualni protok i višak točke, a na početku guranja oduzima se residualni protok i višak točke

```
void guraj(int u, int v)

{

        Console.WriteLine("guraj ({0} => {1})", u, v);

        int kolicinaProtoka =

        Math.Min(visak[u], matricaKapaciteta[u, v] - residualnaMatrica[u, v]) ;

        residualnaMatrica[u, v] += kolicinaProtoka;

        residualnaMatrica[v, u] -= kolicinaProtoka;

        visak[u] -= kolicinaProtoka;

        visak[v] += kolicinaProtoka;

}
```

4.3. Promjeni Visinu

Kod operacije *promjeniVisinu* prvo se deklarira *najmanjaVisina* i postavlja se na najveću moguću vrijednost. Zatim operacija for petljom prolazi kroz sve točke i provjerava jesu li susjedi. Susjede zatim usporedi s varijablom *najmanjaVisina* i ako je ta visina manja postavi tu visinu u varijablu *najmanjaVisina*. Kade operacija prođe sve susjede imat će najmanju visinu susjednu visinu i postavit će točku na kojoj se izvršila operacija na *najmanjaVisina* + 1.

```
void promjeniVisinu(int u)

{

        int najmanjaVisina = int.MaxValue;

        for (int v = 0; v < n; v++)

        {

                if (matricaKapaciteta[u, v] - residualnaMatrica[u, v] > 0)

                {

                        najmanjaVisina = Math.Min(najmanjaVisina, visine[v]);

                        visine[u] = najmanjaVisina + 1;

                }

        }

Console.WriteLine("promjeni visinu tocke {0} u |{2}|", u, (visine[u] - 1), visine[u]);

}
```

4.4. Inicijalizacija

U inicijalizaciji visina izvora se postavlja na dimenziju matrice *n* te višak izvora se postavlja na najveću moguću vrijednost. Zatim for petljom se gura na sve točke koje imaju kapacitet to jest na sve susjede.

```
visine[izvor] = n;

visak[izvor] = int.MaxValue;

Console.WriteLine("promjeni visinu tocke {0} u |{2}|", izvor, (visine[izvor] - 1), visine[izvor]);

for (int i = 0; i < n; i++)

{

        if (matricaKapaciteta[izvor, i] > 0) guraj(izvor, i);

}
```

4.5. Preljev Toka

Preljev toka se koristi na točki s viškom protoka i ponavljat će se dok taj protok ne bude 0. Operacija prvo provjerava sve točke redom pomoću niza *posjecene* i traži moguće guranje. Ako na prvoj točki nije moguće guranje posjecene se inkrementira i gleda se sljedeća točka. To se radi dok se ne pronađe moguće guranje ili dok sve točke nisu posjećene u kojem slučaju se vrši promjena visine. Nakon promjene visine *posjecene* se resetira i opet se ponavlja isti postupak dok na točki više nema viška protoka.

```
void preljevToka(int u)
{
        if (visak[u] > 0) Console.WriteLine("Preljev toka na tocki: " + u);
        else Console.WriteLine("Nema preljeva na točki: " + u);
        while (visak[u] > 0)
        {
                if (posjecene[u] < n)
                {
                        int v = posjecene[u];
                        if (matricaKapaciteta[u, v] - residualnaMatrica[u, v] > 0
                        && visine[u] > visine[v])
                        {
                                guraj(u, v);
                        }
                        else
                        {
                        posjecene[u] += 1;
                        }
                }
                else
                {
                        promjeniVisinu(u);
                        posjecene[u] = 0;
                }
```

4.6. Main

U main-u se deklarira klasa *GT_Algoritam* u kojoj se nalazi metoda *GoldbergTarjan* koja kao argumente prima dvodimenzionalni niz koji predstavlja matricu kapaciteta i dvije točke koje predstavljaju početnu i krajnju točku. Ovdje se pravi matrica kapaciteta i kad se ona preda metodi dobivamo maksimalni protok grafa kojeg ta matrica predstavlja.

```
static void Main(string[] args)
{
        GT_Algoritam algoritam = new GT_Algoritam();
        int[,] matricaKapaciteta = new int[4, 4];
        Console.WriteLine("Ahuja");
        matricaKapaciteta[0, 1] = 2;
        matricaKapaciteta[0, 2] = 4;
        matricaKapaciteta[1, 3] = 1;
        matricaKapaciteta[1, 2] = 10;
        matricaKapaciteta[2, 3] = 5;
        Console.WriteLine("Maksimalni protok mreže: {0} \n", algoritam.GoldbergTarjan
        (matricaKapaciteta, 0, 3));
}
```

4.7. Glavna petlja

Glavna petlja je glavni dio algoritma. Ona određuje na kojim točkama se vrši preljev toka. Prvo se deklarira varijabla indeks. Petlja će se ponavljati dok indeks nije jednak broju točaka koje nisu početak i kraj. Varijabla *staraVisina* pamti visinu trenutne točke i zatim se vrši preljev toka na toj točki. Kada završi preljev toka ako je visina točke veća nego što je bila točka koja na kojoj se vršio preljev se stavlja na početak *listeTocaka* i indeks se resetira na 0. Ovaj postupak se ponavlja do se ne izađe iz while petlje i zatim se vraća konačni rezultat koji je jednak višku protoka na krajnjoj točki.

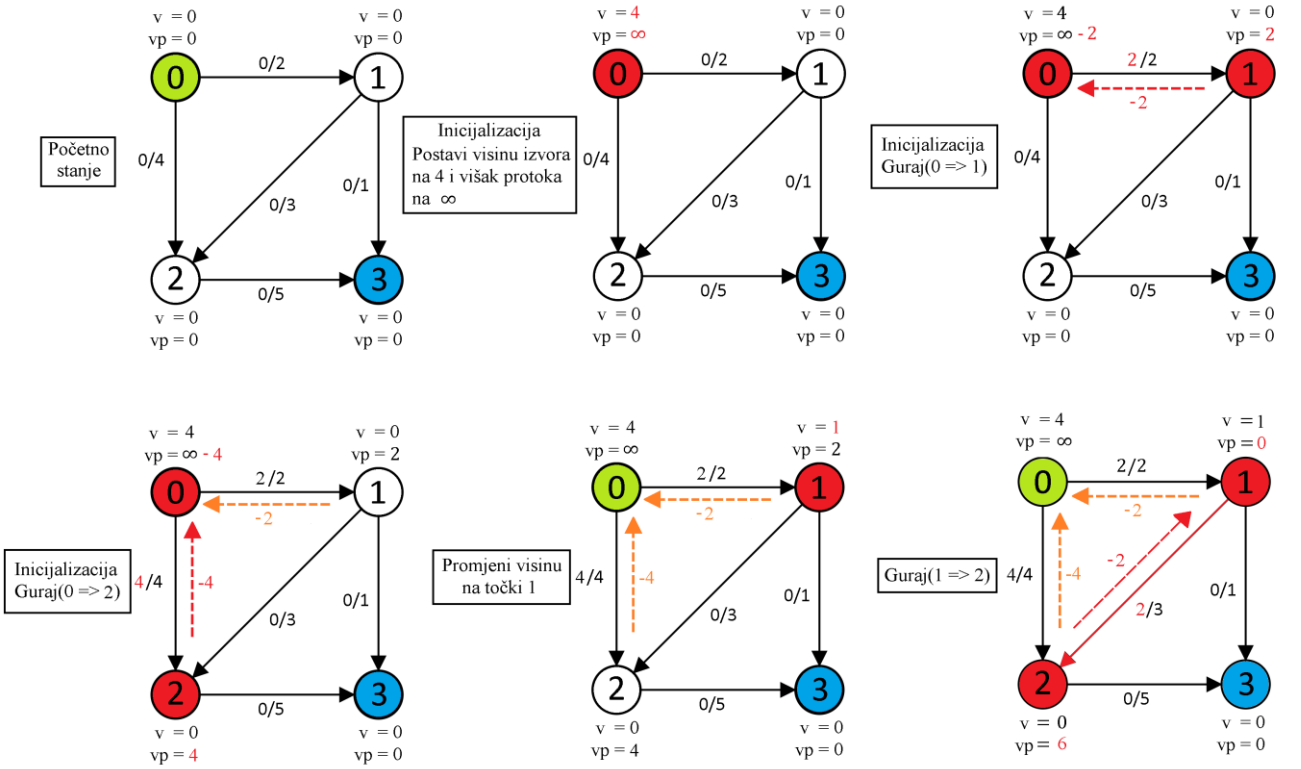
```
int indeks = 0;
while (indeks < listaTocaka.Count)
{
        int u = listaTocaka.ElementAt(indeks);
        int staraVisina = visine[u];
        preljevToka(u);
        if (visine[u] > staraVisina)
        {
                int tocka = listaTocaka.ElementAt(indeks);
                listaTocaka.RemoveAt(indeks);
                listaTocaka.Insert(0, tocka);
                indeks = 0;
        }
        else
        {
                indeks += 1;
        }
}
return visak[kraj];
```

5. PRIMJERI

Ovdje će biti prikazano izvođenje algoritma na danim primjerima.

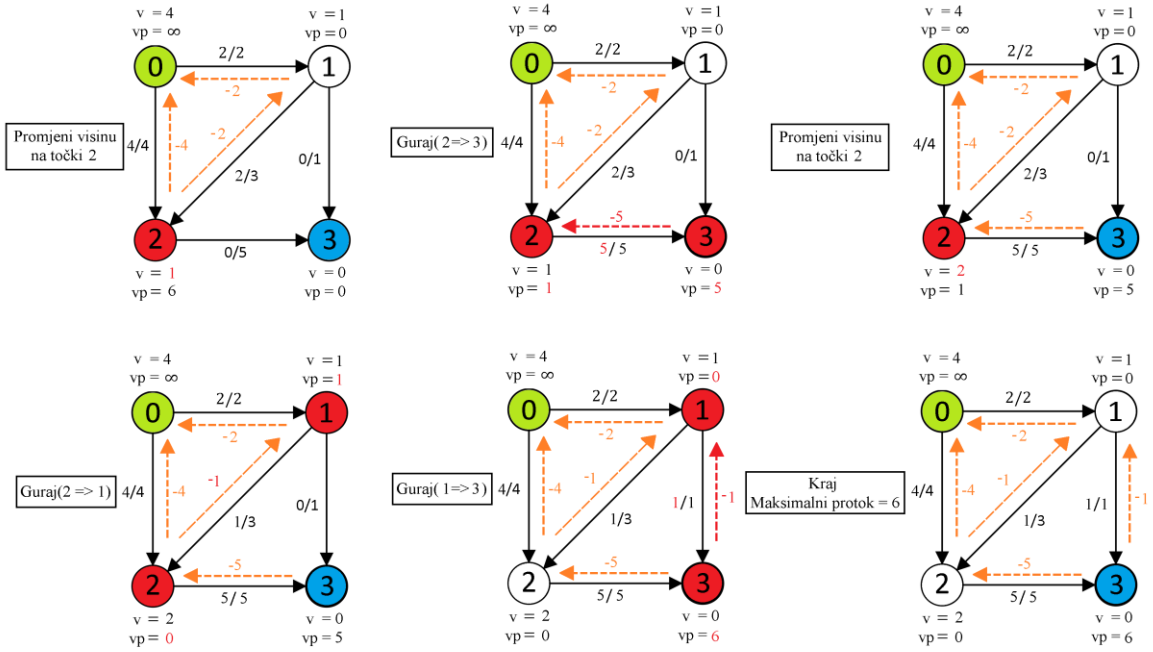
5.1. Ahuja graf

Ovaj graf ima 4 točke i 5 veza između točaka[5]. Prvi korak algoritma je inicijalizacija. Visina početne točke (označena zelenom bojom) postavljamo na 4 i njezin višak protoka na ∞. Poslije toga vrši se guraj operacija na susjedne točke. Sada je završila inicijalizacija i algoritam provjerama može li gurati. U ovom slučaju ne može pa se mora promijeniti visina. Algoritam ima dvije točke koje ispunjavaju uvjete za promjenu visine: točka 1 i 2. Izabire točku jedan zato što je ona prva u listi *listaTocaka.* Visine točke 1 se mijenja na visinu 1 i sada se gura od točke 1 do točke 2.



Slika 5.1. Graf Ahuja – prvi dio

Sada opet nije moguće gurati i mora se promijeniti visina. Točka 2 ima višak protoka pa se njezina visina mijenja iz 0 u 1. Zatim se gura od 2 do 3 i nakon toga se opet mijenja visina na točki 2 iz 1 u 2. Sada guramo od 2 do 1 što je u suprotnom smjeru od usmjerene veze između te dvije točke. To je dopušteno zato što su uvjeti ispunjeni: točke 1 i 2 su susjedi, točka 2 je viša od točke 1 i razlika kapaciteta i residualnog protok tih točaka je veća od 0, ali pošto guramo u suprotnom smjeru sad će se residualni protok povećati s -2 na -1. Poslije toga gura se od 1 do 3. I sada ni jedna točka nema višak protoka osim izvora. Guranje više nije moguće, a ni promjena visine. Sada maksimalni protok ovog grafa jednak je višku protoka na izvoru što je 6 i to je kraj algoritma.



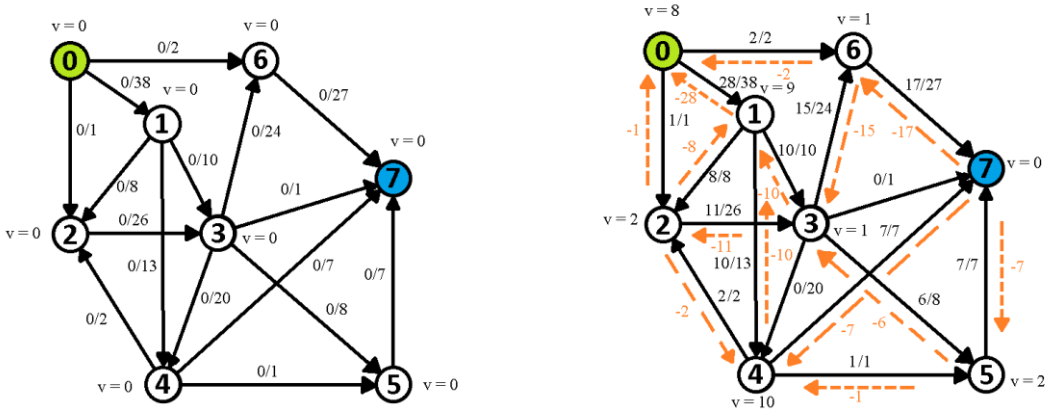
Slika 5.2. Graf Ahuja – drugi dio

5.2. Jungnickel graf

Ovaj graf im 8 točaka [5]. Pošto ovaj graf ima previše koraka za detaljan opis na slici 5.2 se može vidjeti kako će algoritam doći do rješenja, a na slici 5.3 početno i krajnje stanje grafa.

```
Jungnickel
promjeni visinu tocke 0 u |8|
        Guraj (0 => 1)
        Guraj (0 => 2)
        Guraj (0 => 6)
Preljev toka na tocki: 1
        Promjeni visinu tocke 1 u |1|
        Guraj (1 => 2)
        Guraj (1 => 3)
        Guraj (1 => 4)
        Promjeni visinu tocke 1 u |9|
        Guraj (1 => 0)
Nema preljeva na tocki: 1
Preljev toka na tocki: 2
        Promjeni visinu tocke 2 u |1|
        Guraj (2 => 3)
Nema preljeva na tocki: 2
Nema preljeva na tocki: 1
Preljev toka na tocki: 3
        Promjeni visinu tocke 3 u |1|
        Guraj (3 => 4)
Nema preljeva na tocki: 3
Nema preljeva na tocki: 2
Nema preljeva na tocki: 1
Preljev toka na tocki: 4
        Promjeni visinu tocke 4 u |1|
        Guraj (4 => 5)
        Guraj (4 => 7)
        Promjeni visinu tocke 4 u |2|
        Guraj (4 => 2)
        Guraj (4 => 3)
        Promjeni visinu tocke 4 u |10|
        Guraj (4 => 1)
Nema preljeva na tocki: 4
Preljev toka na tocki: 3
        Guraj (3 => 5)
        Guraj (3 => 6)
Preljev toka na tocki: 2
        Promjeni visinu tocke 2 u |2|
        Guraj (2 => 3)
Nema preljeva na tocki: 2
Nema preljeva na tocki: 4
Preljev toka na tocki: 3
        Guraj (3 => 6)
Preljev toka na tocki: 1
        Guraj (1 => 0)
Preljev toka na tocki: 5
        Promjeni visinu tocke 5 u |1|
        Guraj (5 => 7)
        Promjeni visinu tocke 5 u |2|
        Guraj (5 => 3)
Nema preljeva na tocki: 5
Nema preljeva na tocki: 2
Nema preljeva na tocki: 4
Preljev toka na tocki: 3
        Guraj (3 => 6)
Nema preljeva na tocki: 1
Preljev toka na tocki: 6
        Promjeni visinu tocke 6 u |1|
        Guraj (6 => 7)
Nema preljeva na tocki: 6
Nema preljeva na tocki: 5
Nema preljeva na tocki: 2
Nema preljeva na tocki: 4
Nema preljeva na tocki: 3
Nema preljeva na tocki: 1
Maksimalni protok mreze: 31
```

Slika 5.2. Graf Jungnickel – konzola



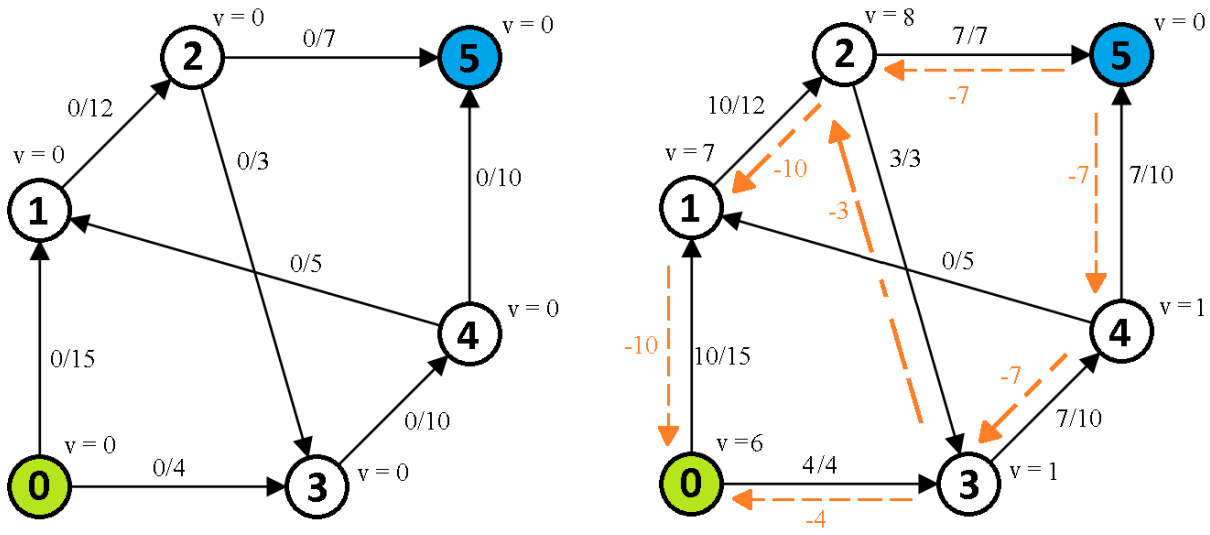
Slika 5.3. Graf Jungnickel početno i krajnje stanje

5.3. Wikipedia graf

Ovaj graf ima 6 točaka i bit će opisan kao i prošli graf [7].

```
Wikipedia
promjeni visinu tocke 0 u |6|
        Guraj (0 => 1)
        Guraj (0 => 3)
Preljev toka na tocki: 1
        Promjeni visinu tocke 1 u |1|
        Guraj (1 => 2)
        Promjeni visinu tocke 1 u |7|
        Guraj (1 => 0)
Nema preljeva na tocki: 1
Preljev toka na tocki: 2
        Promjeni visinu tocke 2 u |1|
        Guraj (2 => 3)
        Guraj (2 => 5)
        Promjeni visinu tocke 2 u |8|
        Guraj (2 => 1)
Nema preljeva na tocki: 2
Preljev toka na tocki: 1
        Guraj (1 => 0)
Preljev toka na tocki: 3
        Promjeni visinu tocke 3 u |1|
        Guraj (3 => 4)
Nema preljeva na tocki: 3
Nema preljeva na tocki: 2
Nema preljeva na tocki: 1
Preljev toka na tocki: 4
        Promjeni visinu tocke 4 u |1|
        Guraj (4 => 5)
Nema preljeva na tocki: 4
Nema preljeva na tocki: 3
Nema preljeva na tocki: 2
Nema preljeva na tocki: 1
Maksimalni protok mreze: 14
```

Slika 5.3. Graf Wikipedia – konzola



Slika 5.4. Graf Wikipedia početno i krajnje stanje

6. ZAKLJUČAK

U ovom radu obrađen je Goldberg-Tarjan algoritam za pronalazak maksimalnog protoka mreže. Algoritam je realiziran u Visual Studio programskom okruženju i napisan u C# programskom jeziku. Zadatak završnog rada je zadovoljen. Operacije algoritma guraj, promjeni visinu i preljev toka su opisane i algoritam je implementiran. Također prikazan je tijek algoritma na 3 grafa: ahuja, jungnickel i wikipedia. Ova verzija algoritma nije najbrža verzija, ali je jedna od jednostavnijih za implementirati. Jedan način na koji bi se ovaj algoritam mogao poboljšati je da na početku izvođenja algoritma uradimo „pretragu u širinu“ ili „*breadth-first search*“ na krajnjoj točki i da postavimo visine svih točaka na njihovu udaljenost od krajnje točke osim izvora čiju visinu postavljamo na visinu jednaku broju točaka. Implementacijom ovoga algoritam bi bio djelotvorniji.

LITERATURA

[1] Protočna mreža, Wikipedija, (23.8.2020.)

https://en.wikipedia.org/wiki/Flow_network

[2] Problem maksimalnog toka, Wikipedija, (23.8.2020.)

https://en.wikipedia.org/wiki/Maximum_flow_problem

[3] Ford-Fulkerson i Edmonds-Karp, cp-algorithms, (23.8.2020.)

https://cp-algorithms.com/graph/edmonds_karp.html

[4] Dinicov algoritam, Wikipedija, (26.8.2020.)

https://en.wikipedia.org/wiki/Dinic%27s_algorithm

[5]Goldberg Tarjan algoritam, adrian-haarbach, (26.8.2020.)

http://www.adrian-haarbach.de/idp-graph-algorithms/implementation/maxflow-push-relabel/index_en.html

[6] Malhotra, Pramodh-Kumar i Maheshwari algoritam, cp-algorithms, (26.8.2020.)

https://cp-algorithms.com/graph/mpm.html

[7] Goldberg Tarjan algoritam, Wikipedija, (27.8.2020.)

https://en.wikipedia.org/wiki/Push%E2%80%93relabel_maximum_flow_algorithm

[8] King Rao Tarjan algoritam, cs.umd, (8.9.2020.)

http://www.cs.umd.edu/~gasarch/BLOGPAPERS/king-rao-tarjan.pdf

[9] James B. Orlin i King, Rao, Tarjan, users.eecs.northwestern, (8.9.2020.)

http://users.eecs.northwestern.edu/~haizhou/457/O(nm)MaxFlow.pdf

SAŽETAK

Ovaj završni rad bavi se općim Goldberg-Tarjan algoritmom i problemom maksimalnog toka. Cilj problema maksimalnog toka je pronaći najveći mogući protok kroz protočnu mrežu. Postoje više algoritama koji se bave tim problemom no ovaj rad obrađuje opći Goldberg-Tarjan algoritam. Algoritam se sastoji 4 operacije guraj, promjena visine, preljev toka i inicijalizacija. Guraj gura višak protoka između točaka, promjeni visinu mijenja visinu točaka kada guranje nije moguće, preljev toka se vrši na točkama s viškom protoka te vrši operacije guraj i promjeni visinu dok preljev toka na toj točki nije nula i inicijalizacija postavlja visinu početne točke na visinu jednaku broju točaka i postavlja višak protoka na toj točki na ∞. Algoritam je implementiran u C# programskom jeziku.

Ključne riječi: protok, protočna mreža, matrica susjedstva, usmjereni graf

ABSTRACT

Title: Goldberg-Tarjan algorithm for finding the max flow of a flow network

This bachelor's thesis deals with the generic Goldberg-Tarjan algorithm and with the maximum flow problem. The goal of the maximum flow problem is to find the biggest flow through the flow through the flow network. There exist many algorithms that deal with the maximum flow problem but this thesis is concerned with the generic Goldberg-Tarjan Algorithm. The algorithm contains 4 operations push, relabel, flow excess and initialization. Push pushes excess flow in between edges, relabel changes the height of the edges when pushing isn't possible, flow excess is done on edges which have excess flow and it uses push and relabel until the excess flow is gone and finally initialization sets the height of the start edge to the numbers od total number of edges and its excess flow to ∞. The algorithm is implemented in the C# programming language.

Key words: flow, flow network, adjecancy matrix, directed graph

ŽIVOTOPIS

Dominik Matković rođen je u Osijeku 11. veljače 1998. Pohađa Osnovnu školu Popovac. 2013. godine upisuje Prvu Srednju Školu Beli Manastir kao tehničar za računalstvo. 2017. godine upisuje preddiplomski sveučilišni studij računarstva na FERIT-u.