

**math.e**

*Hrvatski matematički elektronički časopis*

## **Guraj-Promijeni visinu algoritam za pronalaženje maksimalnog protoka mreže**

optimizacija na grafovima    problem maksimalnog reza

### **Guraj-Promijeni visinu algoritam za pronalaženje maksimalnog protoka mreže**

Mario Zečić  
Ivica Nakić

Problem maksimalnog toka i njemu dualan problem, problem minimalnog reza, iznimno su korisni u modeliranju mnogih problema koji sadrže mreže, bilo cestovne, željezničke, računalne ili društvene. Obično modeliramo tok nekog materijala iz jednog dijela mreže u drugi, a tok mogu biti automobili, vlakovi, bitovi, preporuke i mnogi drugi predmeti i koncepti. Zanimljivo je da su se ova dva problema također pokazala korisnima u problemima modeliranja za koje nije očito da uključuju mreže ili protok materijala. Jedan takav primjer izložiti ćemo u ovom članku.

U ovom članku izložiti ćemo jedan od najefikasnijih poznatih algoritama za rješavanje problema maksimalnog toka, guraj-promijeni visinu algoritam za pronalaženje maksimalnog protoka mreže. Uz teorijsku pozadinu algoritma guraj-promijeni visinu, osvrnut ćemo se i na implementaciju tog algoritma u Javascriptu, kao i neke varijante algoritma koje nude određena poboljšanja.

# 1 Problem maksimalnog toka

## 1.1 Općenito o problemu maksimalnog toka

Prije same definicije problema maksimalnog toka, definirat ćemo pojmove iz teorije grafova koji će nam biti korisni u nastavku.

**Definicija 1.1. Usmjereni graf** je uređeni par  $(G, E)$ , gdje je proizvoljan neprazan konačan skup, a  $E$  je skup bridova. Elemente skupa nazivamo vrhovi, a skupa bridovi.

Kažemo da postoji **put** između vrhova  $s$  i  $t$  ako postoji konačan niz vrhova  $v_0, v_1, \dots, v_n$  takav da vrijedi  $v_0 = s$  i  $v_n = t$ .

Put u kojem se nijedan vrh ne pojavljuje više od jednom nazivamo **jednostavnim putem**. Put koji počinje i završava u istom vrhu zove se **ciklus**. **Jednostavan ciklus** je ciklus u kojem su jedini vrhovi koji se ponavljaju prvi i zadnji.

Definirajmo sada formalno problem maksimalnog toka i pojam toka.

**Definicija 1.2.** Neka je dan usmjereni graf  $G$  te neka su dani kapaciteti bridova  $c_{ij}$  bridova  $(i, j) \in E$ . **s-t tok** je funkcija koja svakom bridu grafa pridružuje nenegativnu vrijednost tako da su sljedeća dva uvjeta zadovoljena:

- za sve bridove  $(i, j) \in E$  vrijedi:  $0 \leq f_{ij} \leq c_{ij}$
- za sve vrhove  $v \in V$  vrijedi da je ukupan tok koji ulazi u vrh  $v$  i jednak ukupnom toku koji izlazi iz vrha  $v$ , točnije:  $\sum_{(i,v) \in E} f_{iv} = \sum_{(v,j) \in E} f_{vj}$

Uz svaki s-t tok definiramo vrijednost toka, koju označavamo  $f$ . To je neto količina toka koja napušta izvor, točnije ukupna količina toka koja izlazi iz izvora smanjena za ukupnu količinu toka koji ulazi u izvor.

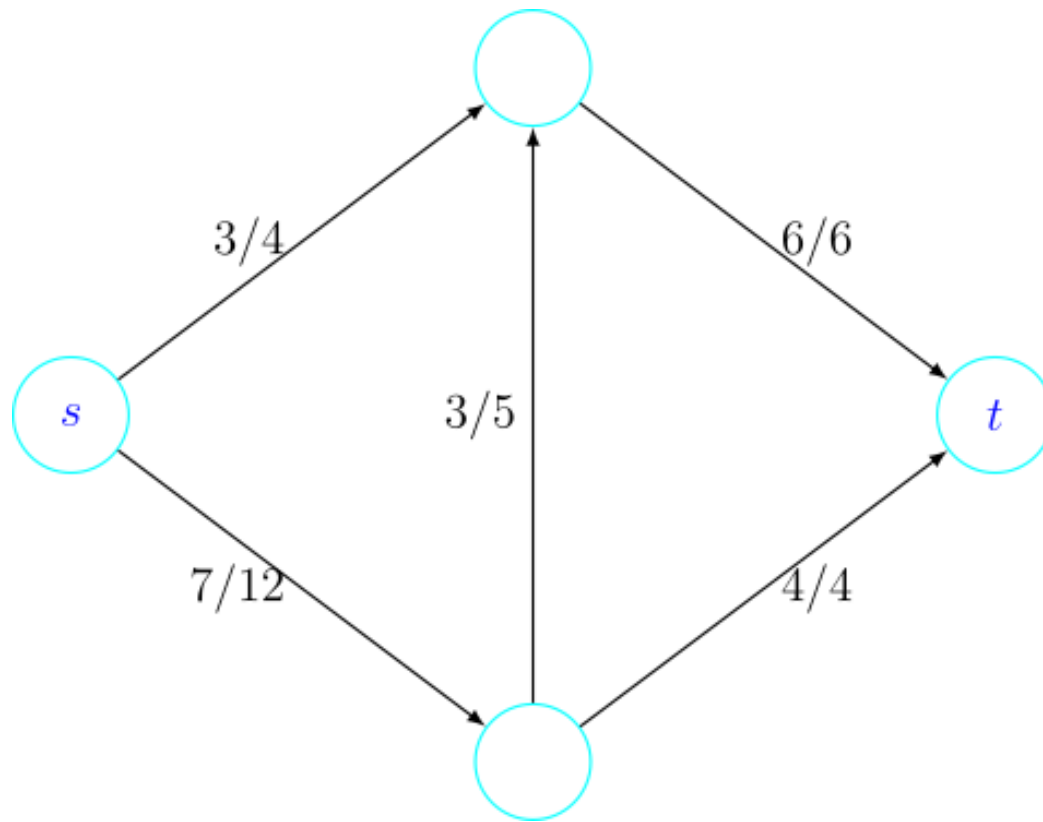
**Definicija 1.3.** Vrijednost s-t toka  $f$  dana je s:

**Definicija 1.4. Problem maksimalnog toka** definiramo ovako: na ulazu su nam dani usmjeren graf  $G$  i kapacitet svakog brida  $c_{ij}$ ,  $c_{ij} \geq 0$ . Uz to su nam dana dva istaknuta vrha  $s, t$ , gdje nazivamo **izvor**, a **ponor**. Cilj je pronaći tok koji maksimizira vrijednost s-t toka  $f$ , dakle veličinu  $|f|$ .

Uvjete  $\sum_{j \in V} f_{ij} = \sum_{j \in V} f_{ji}$  nazivamo **ograničenja kapaciteta**, a **ograničenja očuvanja toka**.

Možemo se pitati zašto se vrijednost toka računa kao neto količina toka koji izlazi iz izvora, a ne kao neto količina toka koji ulazi u ponor, ali može se vidjeti da su te dvije vrijednosti uvijek jednake zbog ograničenja očuvanja toka.

Sada vidimo da je cilj problema maksimalnog toka pronaći s-t tok koji maksimizira  $|f|$ . Primijetimo da je, zato što su vrijednosti kapaciteta nenegativne, tok za kojeg vrijedi  $|f| = \sum_{j \in V} f_{ij}$  s-t tok. Tada slijedi da je vrijednost maksimalnog toka uvijek nenegativna. Na slici 1 možemo vidjeti jedan primjer maksimalnog toka u grafu.



**Slika 1:** Prikaz jednog maksimalnog toka u zadanom grafu. Notacija 'x/y' pored svakog brida označava da taj brid ima kapacitet y, a količina toka na tom bridu je x.

Iako definicija koju smo dali daje dobro postavljen problem, uvodimo novu definiciju pojma toka koja će nam pojednostavniti dokaze. U novoj definiciji uzimamo u obzir samo gornju granicu na vrijednost toka, umjesto gornju i donju kao u definiciji koju smo ranije dali. Svakom bridu uvodimo dodatan, njemu suprotan brid s uvjetom da ako vrijednost toka na bridu iznosi  $x$ , tada vrijednost toka na bridu iznosi  $x$ . Formalno, novi graf možemo biti multigraf, graf koji dozvoljava više bridova s istim vrhovima. Ovaj dodatan uvjet nazivamo **iskrivljenom simetrijom**. Sada se donja ograda može zamijeniti s uvjetom  $\sum_{e \in E} f(e) = 0$ , jer tada povlači:  $\sum_{e \in E} f(e) = 0$ .

Nadalje, pod novom definicijom, suma vrijednosti svih tokova na bridovima koji izlaze iz vrha jednaka je neto vrijednosti toka koji izlazi iz čvora po prethodnoj definiciji. Točnije, ako s označimo novi skup bridova (uključujući suprotne bridove), vrijedi:

Tada ograničenje očuvanja toka postaje:

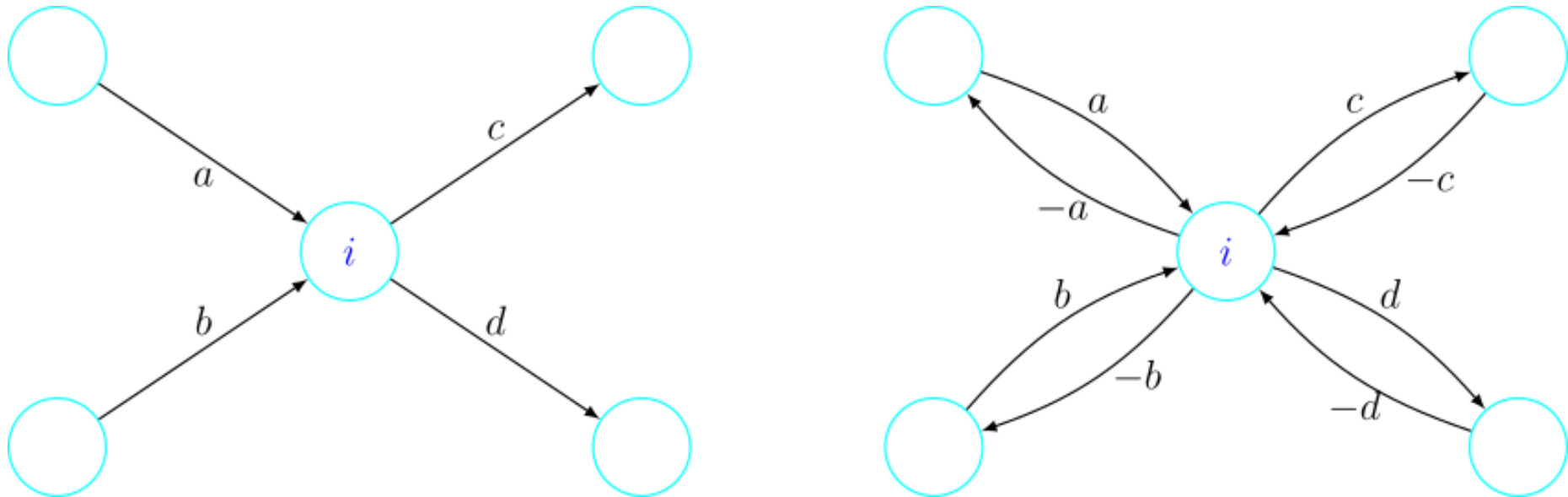
Slika 2 prikazuje razliku između ograničenja očuvanja toka po staroj i novoj definiciji. Slično se može vidjeti

da vrijednost toka iznosi  
sada tok formalno:

Radi jednostavnosti, od sada ćemo novi skup bridova označavati s . Definirajmo

**Definicija 1.5. s-t tok** je funkcija koja svakom bridu pridružuje vrijednost tako da vrijedi:

- $f(e)$  je ukupna vrijednost toka na bridovima koji izlaze iz vrha jednaka 0, točnije
- za sve  $v$  je ukupna vrijednost toka na bridovima koji ulaze u vrh jednaka 0, točnije

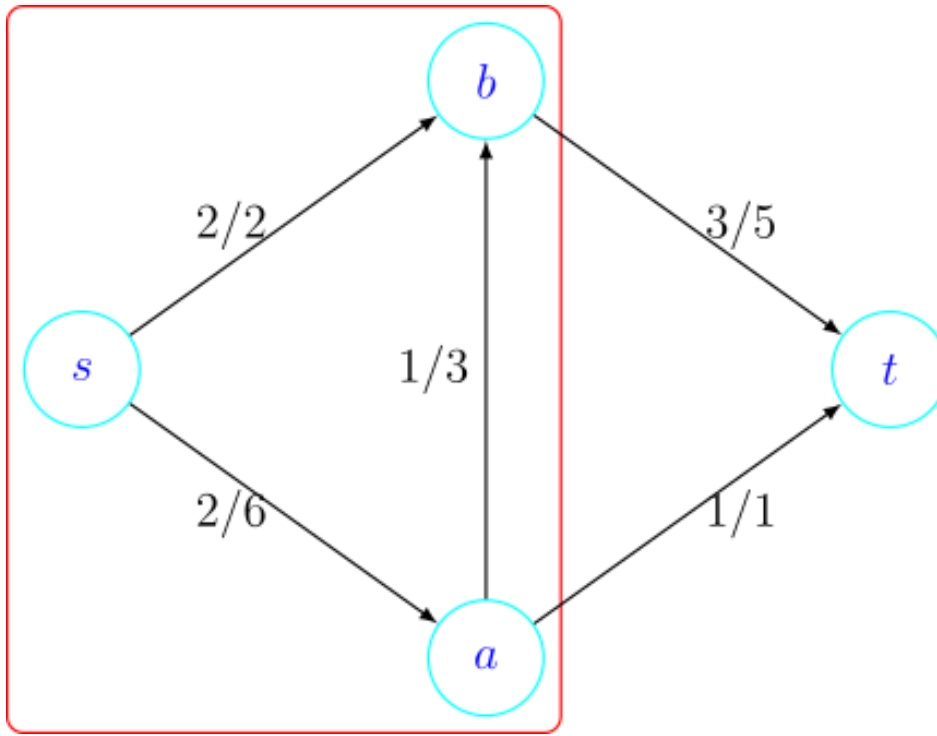


**Slika 2:** Na lijevom grafu prikazano je očuvanje toka po definiciji 1.2, dok je na desnom grafu prikazano očuvanje toka po definiciji 1.5. Vidimo da je neto vrijednost toka u vrhu jednaka  $f(i)$ , a vrijednost toka koji izlazi iz vrha jednaka  $f(i)$ .

## 1.2 Uvjeti optimalnosti

Nakon što smo definirali tok i problem maksimalnog toka, pitamo se kako odrediti je li dani tok maksimalan ili nije. Promotrimo prvo jedan primjer.

**Primjer 1.6.** Promotrimo dani graf. Je li označeni tok maksimalan?



Po definiciji vrijednosti toka vidimo da je  $1$ . Također vidimo da svaka jedinica toka koja putuje od vrha do vrha mora proći bridovima koji idu iz crvenog područja u područje koje nije crveno. Iz toga zaključujemo da maksimalan tok ne može biti veći od  $1$ . Takvim razmišljanjem dolazimo do ideje minimalnog reza.

Formalizirajmo sada intuiciju iz primjera.

**Definicija 1.7.** *s-t rez* je podskup vrhova  $U$  takav da je  $s \in U$  i  $t \notin U$ .  
 Za bridove koji izlaze iz  $U$  kažemo da su u rezu definiranom  $s$ , formalno:  $E^+(U)$ . Analogno definiramo skup svih  
 bridova koji ulaze u  $U$ :  $E^-(U)$ . **Kapacitet** s-t reza je suma kapaciteta svih bridova koji izlaze iz  $U$ , točnije:

Sada, kad smo formalno definirali pojam reza, možemo dokazati da za svaki s-t tok  $f$  i s-t rez  $U$  vrijedi da je vrijednost toka manja ili jednaka kapacitetu reza  $U$ .

**Lema 1.8.** Za svaki s-t tok  $f$  i s-t rez  $U$  vrijedi:

*Dokaz.* Korištenjem definicije vrijednosti foka i ograničenja očuvanja toka za sve (prisjetimo se da vrijedi  $f(u,v) \leq c(u,v)$ ) imamo:

Sada možemo rastaviti sumu na dva dijela ovisno o tome je li početni vrh brida element skupa ili nije. Primijetimo da, ako su  $u, v \in S$ , suma sadržava  $f(u,v)$ , pa se ta dva sumanda poništavaju. Sada korištenjem ograničenja kapaciteta imamo:  $\square$

**Korolar 1.9.** *Neka je  $s$ - $t$  tok i  $s$ - $t$  rez. Tada je  $f = c$  ako i samo ako je  $f = c$ .*

*Dokaz.* Ako je  $f = c$ , onda je posljednja nejednakost u prijašnjem dokazu jednakost, pa je time ova implikacija dokazana.

Ako je  $f = c$ , onda mora vrijediti  $f(u,v) = c(u,v)$ . Međutim, budući da su svi sumandi u obje sume nenegativni i budući da vrijedi  $f(u,v) \leq c(u,v)$ , zaključujemo da tada mora vrijediti  $f(u,v) = c(u,v)$ .  $\square$

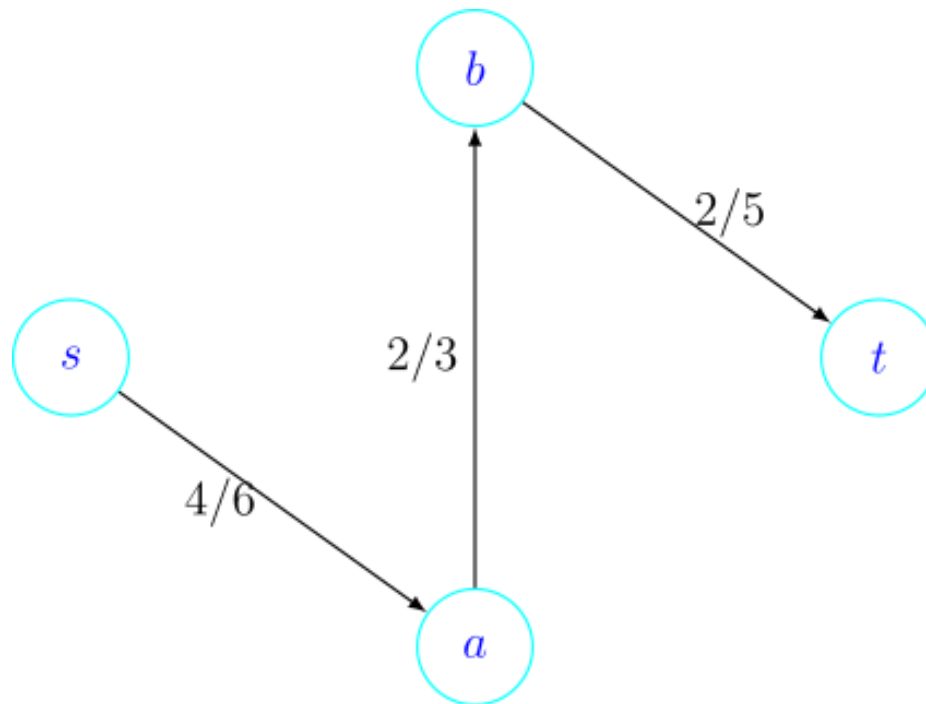
Sada možemo vidjeti da dani tok u grafu iz primjera 1.6 nije maksimalan jer je skup  $S$   $s$ - $t$  rez, a ne vrijedi  $f = c$ , što je u kontradikciji s prijašnjim korolarom.

**Definicija 1.10. Minimalni  $s$ - $t$  rez** je rez s minimalnim kapacitetom, to jest:

Po lemi 1.8 zaključujemo da je vrijednost maksimalnog toka manja ili jednaka kapacitetu minimalnog reza. Kada bi u primjeru 1.6 povećali tok na bridovima  $u_1v_1$  i  $u_2v_2$  za 2, dobili bismo tok čija je vrijednost jednaka kapacitetu minimalnog reza za  $C$ . To nas dovodi do jednog od najistaknutijih rezultata u teoriji mrežnih tokova.

**Teorem 1.11 (Ford, Fulkerson).** *Vrijednost maksimalnog  $s$ - $t$  toka jednaka je vrijednosti minimalnog  $s$ - $t$  reza.*

Prije navođenja idućeg važnog teorema, definirajmo još nekoliko bitnih pojmova. Ako nam je dan tok u grafu  $G$ , definiramo **rezidualni kapacitet**  $r(u,v)$ . Primijetimo da je rezidualni kapacitet uvijek nenegativan zbog ograničenja kapaciteta. Bridove  $e$  čiji je rezidualni kapacitet 0 nazivamo **zasićenim** bridovima. **Rezidualni graf** u odnosu na tok definiramo kao graf  $G_f$  gdje je



**Slika 3:** Rezidualni graf grafa iz primjera 1.6. Vidimo da je  $s \rightarrow a \rightarrow b \rightarrow t$  jedan povećavajući put.

Primjetimo da za suprotne bridove vrijedi:  $c_{ij} = c_{ji}$  pa zaključujemo da rezidualni kapacitet suprotnog brida odgovara tome koliko maksimalno možemo smanjiti tok na bridu  $i \rightarrow j$ . Kao primjer rezidualnog grafa dana je slika 3.

Ako postoji put od vrha do vrha u rezidualnom grafu takav da su svi bridovi na tom putu elementi skupa  $E^+$ , onda takav put nazivamo **povećavajući put**. Jedan takav put može se vidjeti na slici 3. Kako svi bridovi u povećavajućem putu imaju strogo pozitivan rezidualni kapacitet, zaključujemo da tok nije maksimalan. Naime, možemo dobiti tok čija je vrijednost veća od toka tako da povećamo tok po bridovima koji se nalaze na povećavajućem putu.

Formalno, neka je najmanji rezidualni kapacitet na povećavajućem putu  $\delta$ , točnije  $\delta = \min_{(i,j) \in E^+} c_{ij}$ . Primjetimo da, jer su rezidualni kapaciteti svih bridova na povećavajućem putu strogo pozitivni, vrijedi  $\delta > 0$ . Tada definiramo novi tok ovako:  $f' = f + \delta \cdot \chi_P$  č Nekada kažemo da gurnemo jedinica toka po putu  $P$ , rezultirajući tokom  $f'$ . Također za

nekada kažemo da je rezidualni kapacitet puta  $\delta$ .

Preostalo je još pokazati da je tok  $f'$ . Možemo, bez smanjenja općenitosti, pretpostaviti da je jednostavan put. Uvjeti očuvanja i dalje vrijede za sve vrhove izvan puta  $P$ . Po definiciji od  $\delta$ , za sve  $(i,j) \in E^+$ , vrijedi:  $c_{ij} - f'_{ij} = c_{ij} - f_{ij} - \delta$  pa je time dokazano da je zadovoljeno ograničenje kapaciteta.

Iskrivljena simetrija očito vrijedi za sve bridove koji nisu u  $E^+$ , a za  $(i,j) \in E^+$  vrijedi:



Još je preostalo pokazati da je zadovoljeno ograničenje očuvanja toka. To svojstvo je očito zadovoljeno za sve vrhove koji nisu dio puta. Ako je vrh na putu, onda postoje bridovi koji su na putu, pa za njih vrijedi  $f(u,v) \leq c(u,v)$  i  $f(v,w) \leq c(v,w)$ . Tada imamo: budući da je tok te za njega vrijedi svojstvo ograničenja očuvanja toka. Slično, ako je prvi brid na putu, onda je  $f(u,v) \leq c(u,v)$ , pa budući da je  $f(v,w) \leq c(v,w)$  vrijedi:

Dakle je veći tok od  $f$ . Sada možemo dokazati teorem čija je direktna posljedica teorem 1.11.

**Teorem 1.12.** *Iduće 3 tvrdnje ekvivalentne su za svaki s-t tok :*

1. je maksimalan tok;
2. ne postoji povećavajući put u ;
3. , za neki s-t rez .

*Dokaz.* : Ovu implikaciju smo već dokazali, točnije njen obrat po kontrapoziciji, prije navođenja samog teorema kad smo dokazali da ako postoji povećavajući put u , onda nije maksimalan.

: Neka je skup svih vrhova do kojih postoji put iz vrha u takav da je rezidualan kapacitet svakog brida na tom putu strogo pozitivan. Jer ne postoji povećavajući put u , znamo da vrijedi  $f(u,v) = c(u,v)$ . Također, za svaki brid takav da je  $f(v,w) < c(v,w)$ , mora vrijediti da je  $f(u,v) < c(u,v)$ , to jest  $f(u,v) < c(u,v)$ . Tada, po korolaru 1.9, zaključujemo da je  $f(u,v) < c(u,v)$ .

: Po lemi 1.8 znamo da vrijedi  $f(u,v) = c(u,v)$ , za svaki s-t tok i s-t rez . Jer za neki s-t tok i s-t rez vrijedi  $f(u,v) < c(u,v)$ , zaključujemo da taj mora biti maksimalan tok, a minimalan rez.  $\square$

Teorem 1.12 daje nam ideju za algoritam za pronalaženje maksimalnog toka: krenemo od toka  $f$ , tražimo povećavajući put te ažuriramo tok onako kako smo opisali u dokazu teorema 1.12. Problem ovog algoritma je da, ako ne biramo povećavajuće puteve pažljivo, algoritam nije polinomijalan.

**Algoritam 1:** Osnovni algoritam povećavajućeg puta za pronalaženje maksimalnog toka

```
dok postoji povećavajući put u :  
  Gurni tok u  
  Ažuriraj  
Vrati ;
```

Međutim, ovaj nas algoritam vodi do zanimljivog zaključka: ako su svi kapaciteti cijeli brojevi, tada postoji maksimalan tok takav da je  $f(u,v) = c(u,v)$  i algoritam 1 pronalazi jedan takav tok. Ako su sve vrijednosti cijeli brojevi, tada za tok kažemo da je **cijeli**. Lako je vidjeti da ova tvrdnja vrijedi. Naime, jer su na početku sve vrijednosti jednake 0 i sve vrijednosti cijeli brojevi, zaključujemo da su svi rezidualni kapaciteti cijeli brojevi. Tada slijedi da je cijeli broj, pa onda i za novi tok vrijedi da je svaka vrijednost cijeli broj.

**Propozicija 1.13 (Svojstvo cijelosti).** Ako su svi kapaciteti cijeli brojevi, tada postoji maksimalan tok čija je vrijednost također cijeli broj.

Označimo s maksimalan kapacitet, točnije . Tada možemo ograničiti broj iteracija (to jest broj povećavajućih puteva koje moramo pronaći) osnovnog algoritma s . Označimo s broj bridova, a s broj vrhova grafa. Naime, jer je maksimalan tok neto vrijednost toka koji izlazi iz izvora, u najgorem slučaju imamo bridova koji izlaze iz izvora takvi da su svi kapaciteta , tako da je najveća moguća vrijednost maksimalnog toka . Ako su kapaciteti cijeli brojevi, tada se vrijednost toka u svakoj iteraciji poveća barem za 1, pa iz toga zaključujemo da je maksimalan broj iteracija . Povećavajući put u grafu možemo pronaći u vremenu , pa je ukupna složenost osnovnog algoritma jednaka .

Iako je vrijednost dio ulaza, ovaj algoritam nije polinomijalan u veličini ulaza problema jer pretpostavljamo da nam je ulaz dan u binarnom zapisu. Broj bitova potreban da bi se zapisao broj u binarnom zapisu je najviše . Tada je eksponencijalan u odnosu na veličinu zapisa broja . Tada za algoritam 1 kažemo da je **pseudopolinomijalan**. U nastavku ćemo pronaći i analizirati neke polinomijalne algoritme za pronalaženje maksimalnog toka.

### 1.3 Zajedničko korištenje automobila

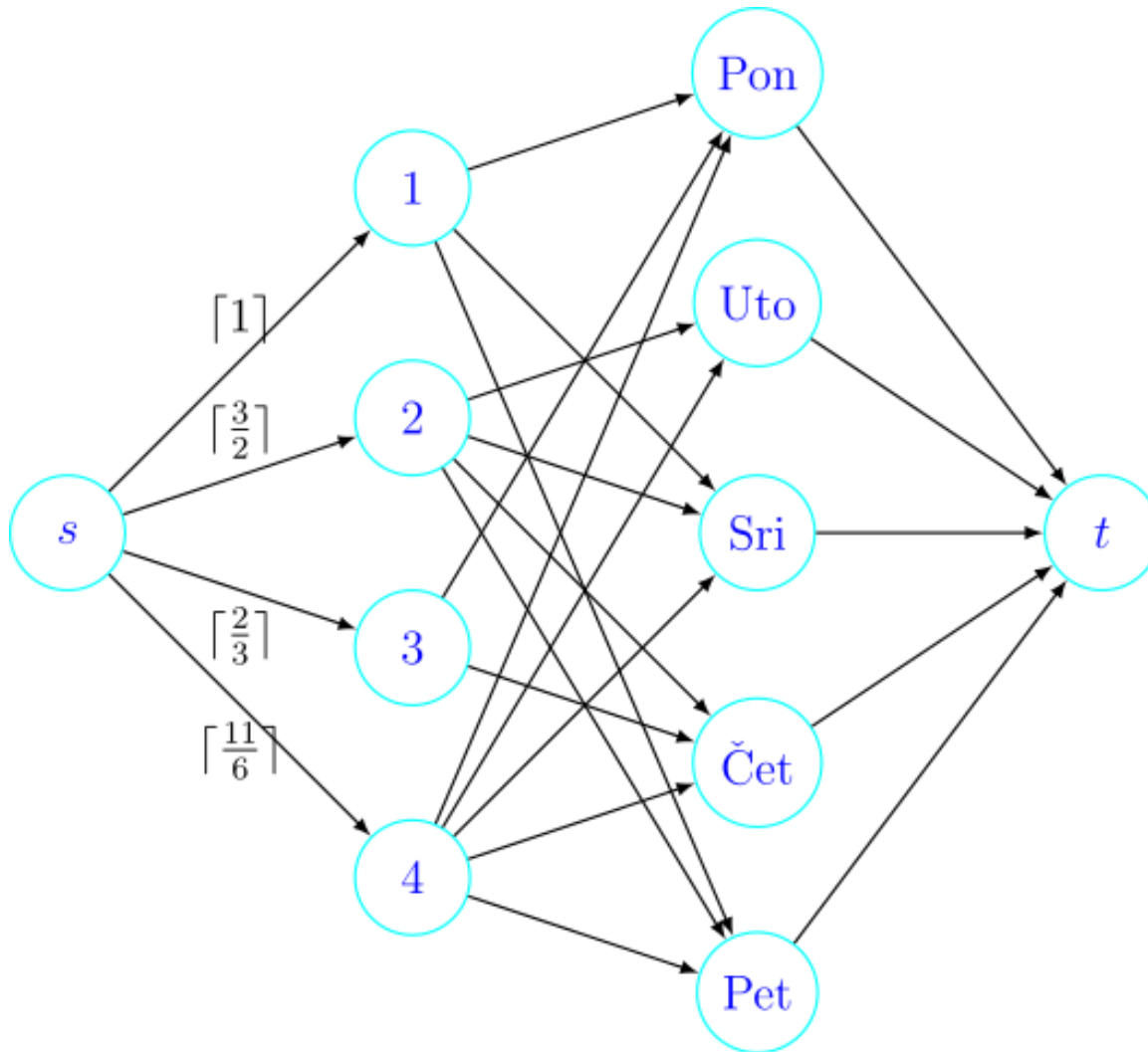
Prije nego što krenemo tražiti polinomijalne algoritme za rješavanje problema maksimalnog toka, promotrimo jedan primjer primjene na problem koji na prvi pogled ne uključuje ni tok ni mrežu.

Primjer koji promatramo uključuje pravednu raspodjelu vozačkih odgovornosti za skupinu ljudi koji zajedno voze. Svaki tjedan ljudi koji se zajedno voze obavijeste ostale kojim danima će koristiti automobil. Oni bi htjeli pravedno rasporediti odgovornost vožnje i tako dolaze do iduće ideje: na dan kada se ljudi vozi u automobilu, svaki od njih dobiva udjela odgovornosti vožnje. Neka je ukupni udio odgovornosti -te osobe za taj tjedan. Onda zaključujemo da bi -ta osoba u tom tjednu trebela voziti najviše dana. U idućoj tablici možemo vidjeti jedan primjer s četvero ljudi.

	Pon	Uto	Sri	Čet	Pet
1	X		X		X
2		X	X	X	X
3	X			X	
4	X	X	X	X	X

Vidimo da osobe 1, 3 i 4 dobiju odgovornosti za ponedjeljak, osobe 2, 3 i 4 odgovornosti za utorak, osobe 1, 2 i 4 odgovornosti za srijedu i petak te osobe 2, 3 i 4 odgovornosti za četvrtak. Tada vidimo da vrijedi  $x_{1,1} = 1$ ,  $x_{2,2} = 1$ ,  $x_{3,3} = 1$  i  $x_{4,4} = 1$ . Primijetimo da je  $x_{i,j} = 1$  jer je suma odgovornosti za svaki dan jednaka 1.

Sada ćemo postaviti instancu problema maksimalnog toka čijim rješavanjem ćemo dobiti tko treba voziti koji dan. Da bismo to napravili, treba nam graf čiji skup vrhova sadrži, osim izvora i ponora, po jedan vrh za svaku osobu i po jedan vrh za svaki radni dan u tjednu. Zatim dodamo po jedan brid kapaciteta od izvora do vrha koji predstavlja osobu te po jedan brid kapaciteta 1 od vrha koji predstavlja neki dan u tjednu do ponora. Za kraj dodamo brid kapaciteta 1 od osobe do svakog dana u kojem ta osoba koristi automobil.



**Slika 4:** Instanca problema maksimalnog toka za primjer zajedničkog korištenja automobila danog gore. Svi neoznačeni bridovi su kapaciteta 1.

Tvrdimo da ako u mreži postoji cijeli tok vrijednosti 5, tada možemo raspodijeliti odgovornosti vožnje osobama. Primijetimo da rez ima kapacitet 5 (po jedan brid kapaciteta 1 za svaki radni dan), pa ako takav tok postoji, onda je i maksimalan. Zato je svaki brid od nekog dana do ponora zasićen i vrijednost toka na tim bridovima je 1. Za svaki vrh koji predstavlja dan, zbog ograničenja očuvanja toka i činjenice da je cijeli, vrijedi da postoji zasićeni brid od vrha koji predstavlja neku osobu do tog dana. Tada odgovornost vožnje za neki dan postavljamo na osobu od koje dolazi brid koji je zasićen. Tada osoba može voziti najviše dana (zbog ograničenja očuvanja toka, ograničenja kapaciteta i činjenice da je kapacitet brida od izvora do čvora jednak), što smo i htjeli.

Sada je još preostalo pokazati da mreža sadrži cijeli tok vrijednosti 5. Međutim, zbog [svojstva cijelosti](#), dovoljno je pokazati da postoji bilo kakav tok vrijednosti 5, pa će činjenica da su svi kapaciteti cijeli brojevi povlačiti da postoji cijeli tok vrijednosti 5. Tok odgovarat će raspodijeli odgovornosti koju smo opisali na početku: ako u nekom danu osoba koristi automobil, svakoj osobi dati ćemo odgovornosti. To znači da ćemo postaviti tok brida koji spaja osobu s danom na . Jer je suma tokova koji ulaze u vrh koji predstavlja neki dan jednaka 1, a vrijednost toka koji izlazi iz vrha koji predstavlja osobu jednak , znamo da smo dobili tok čija je vrijednost .

Ovaj primjer dobro ilustrira moć [svojstva cijelosti](#). Naime, bilo nam je dovoljno pronaći neki maksimalan tok u grafu čiji su kapaciteti cijeli brojevi da bismo zaključili da postoji cijelobrojan maksimalan tok.

Jedno rješenje problema, dobijeno korištenjem implementacije opisane u sekciji 3, je da vozač 1 vozi petkom, vozač 2 srijedom, vozač 3 ponedjeljkom te vozač 4 utorkom i četvrtkom. Ova raspodjela je pravedna jer -ti vozač vozi najviše dana u tjednu.

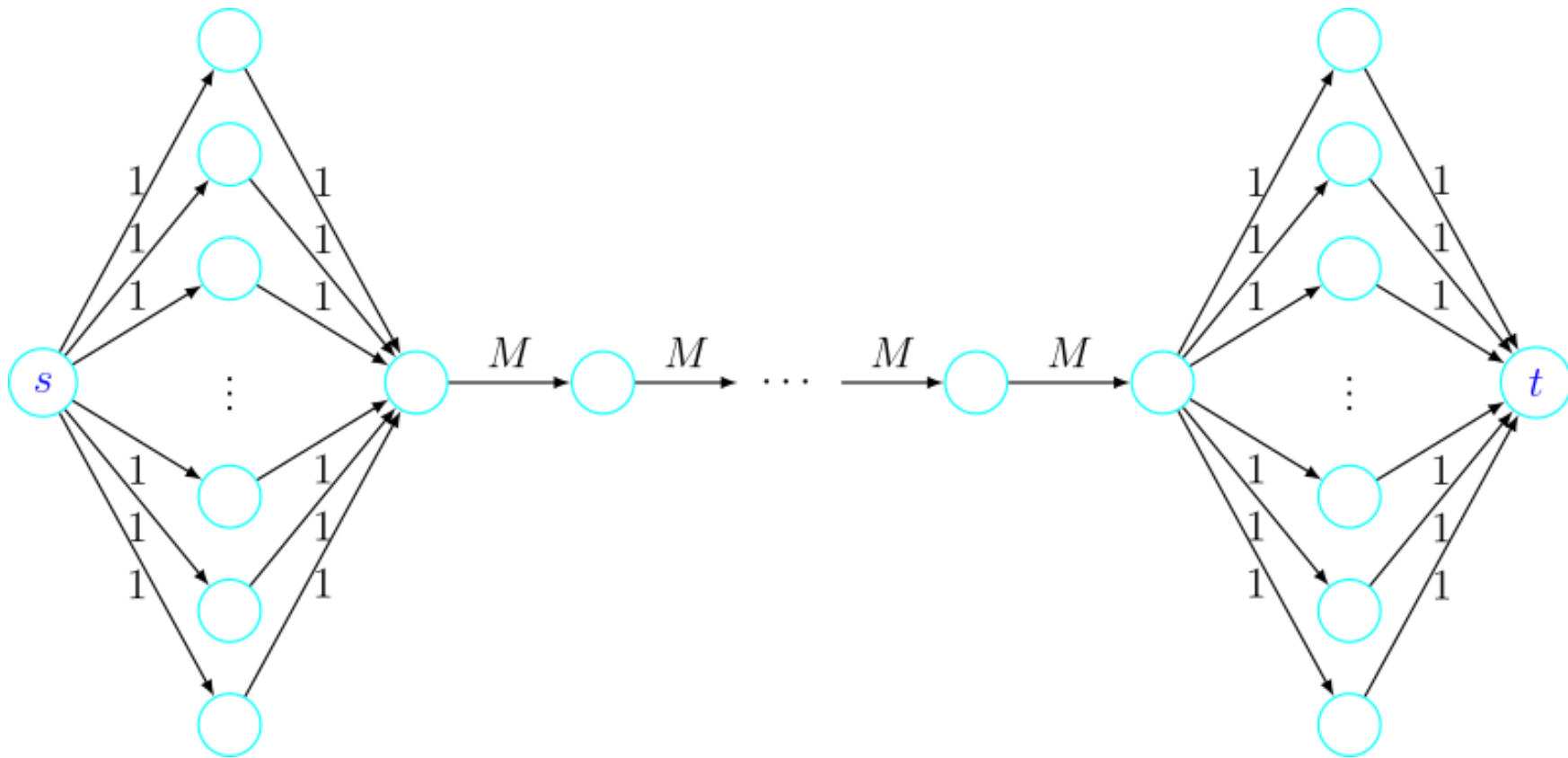
## 2 Guraj-promijeni visinu algoritam

### 2.1 Opis algoritma

U ovoj sekciji promotrit ćemo algoritam koji je idejno poprilično drugačiji od [osnovnog algoritma za pronalaženje maksimalnog toka](#), ali se u praksi pokazao među najbržim poznatima.

Jedan od problema osnovnog algoritma ilustriran je [slikom 5](#). Taj graf sadrži vrhova desno od izvora, vrhova lijevo od ponora i jako dugačak put između ta 2 skupa vrhova. Po osnovnom algoritmu, svaki put kada povećamo vrijednost puta, povećamo ju za 1 i to povećanje šaljemo dugim putem. Bilo bi bolje kada bismo nekako mogli povećati put za odjednom, stoga bismo onda morali samo jednom slati novi tok po dugačkom putu. Algoritam koji ćemo promatrati u ovoj sekciji, **guraj-promijeni visinu algoritam**, moći će napraviti upravo to. Dok smo kod osnovnog algoritma održavali tok i povećavali ga dok nismo pronašli s-t rez kapaciteta jednakog vrijednosti toka, guraj-promijeni visinu algoritam radi upravo suprotno: u njemu održavamo s-t rez sve dok ne dobijemo tok vrijednosti jednake kapacitetu s-t reza .

Guraj-promijeni visinu algoritam održava posebnu vrstu toka koji nema sva svojstva kao i tok kojeg smo definirali. Tu vrstu toka nazivamo **predtok**. Svojstvo koje predtok ne zadovoljava je svojstvo očuvanja toka: umjesto da zahtijevamo da je neto vrijednost toka za svaki vrh jednaka 0, zahtijevamo samo da je nenegativna. Tada neto vrijednost toka u vrhu nazivamo **višak** u vrhu i označavamo s .



**Slika 5:** Instanca problema maksimalnog toka kod kojeg osnovni algoritam ima loše performanse.

**Definicija 2.1.**  $s$ - $t$  **predtok** je funkcija  $f$  takva da vrijedi:

- $f(u, v) \leq c(u, v)$  ;
- tok koji ulazi u vrh  $v$  je nenegativan, točnije  $f(u, v) \geq 0$  ;
- $f(u, v) = f(v, u)$  .

**Višak** od za predtok definiramo ovako:  $excess(v) = \sum_{u \in V} f(u, v) - \sum_{u \in V} f(v, u)$  .

Uz održavanje predtoka, guraj-promijeni visinu algoritam održava i oznake visine za svaki vrh  $v \in V$  . Ako je  $h(v) = 0$  , gdje je broj vrhova, onda je oznaka visine donja ograda na (netežinsku) udaljenost od vrha do ponora po bridovima pozitivnog rezidualnog kapaciteta, tj. po bridovima rezidualnog grafa.

**Definicija 2.2.** Oznaka visine je **valjana**, za neki  $v$  i predtok  $p$ , ako vrijedi:

- $v \geq 0$ ;
- $v \leq c_{ij}$ ;
- $v \leq p_i - p_j$ .

Jedna posljedica postojanja valjane oznake visine je da za svaki predtok postoji s-t rez u rezidualnom grafu, to jest ne postoji povećavajući put u rezidualnom grafu. Dokaz je dan sljedećom lemom 2.3

**Lema 2.3.** Za dani predtok  $p$  i valjanu oznaku visine  $v$  ne postoji povećavajući put u rezidualnom grafu.

*Dokaz.* Pretpostavimo da postoji jednostavan put od vrha do vrha takav da za svaki brid na putu vrijedi  $v$ . Budući da je jednostavan put, može sadržavati najviše  $m$  bridova, a budući da je valjana oznaka visine vrijedi  $v \leq c_{ij}$ . Znamo da vrijedi  $v \leq p_i - p_j$ , pa budući da je valjana oznaka visine mora vrijediti:  $v \leq p_i - p_j$ , što je kontradikcija s definicijom valjane oznake visine.  $\square$

Sada vidimo da je cilj guraj-promijeni visinu algoritma postepeno pretvarati predtok u tok i pri tome održavati valjanu oznaku visine. Jednom kad imamo tok  $f$  i valjanu oznaku visine  $v$  za po lemi 2.3 znamo da ne postoji povećavajući put u rezidualnom grafu, pa je tok maksimalan.

Glavna ideja pretvaranja predtoka u tok je postepeno gurati što je više moguće pozitivnog viška prema ponoru. Tok ćemo gurati po bridovima za koje mislimo da su na najkraćem putu do ponora. Ako vrh ima pozitivan višak  $v$  i postoji brid  $(i, j)$  s pozitivnim rezidualnim kapacitetom, možemo dobiti novi predtok ako povećamo tok na bridu  $(i, j)$ : povećavanje vrijednosti smanji višak u vrhu  $i$ , ali ga poveća u vrhu  $j$ . Da bismo osigurali da guramo tok po bridu koji je na najkraćem putu od  $i$  do ponora, guramo samo po bridovima za koje vrijedi  $v \leq p_i - p_j$ , to jest je bliže ponoru nego  $j$ .

Uvodimo terminologiju koju ćemo često koristiti u daljnjim razmatranjima: kažemo da je brid **dopustiv** ako je  $(i, j)$  i  $v \leq p_i - p_j$ . Tok ćemo povećavati samo po dopustivim bridovima. Kako želimo gurnuti što je više toka moguće po dopustivim bridovima, gurat ćemo minimum između viška i rezidualnog kapaciteta  $\min\{v, c_{ij} - f_{ij}\}$ .

Što ako imamo vrh  $i$  sa strogo pozitivnim viškom, ali nemamo dopustivih bridova, to jest vrijedi  $v > p_i - p_j$  takve da vrijedi  $v \leq p_i - p_j$ ? U tom slučaju, intuicija nam govori da nam je oznaka visine za vrh preniska. Naime, svi vrhovi takvi da postoji dopustiv brid imaju oznaku visine barem onoliko koliko ima vrh  $i$ , pa onda zaključujemo da mora imati veću oznaku visine nego što sad ima. Zato ćemo, u slučaju da ne postoji dopustiv brid iz vrha  $i$  s pozitivnim viškom, **promijeniti visinu** vrha  $i$ : postaviti ćemo da bude jednak minimumu vrijednosti  $v$ , za sve bridove  $(i, j)$ . Time ćemo dobiti barem jedan dopustiv brid iz vrha  $i$  s pozitivnim viškom, a oznaka visine ostat će valjana. Primijetimo da ako vrh  $i$  ima pozitivan višak, onda je  $v > p_i - p_j$ , za neki brid  $(i, j)$ , što onda povlači da postoji barem jedan brid koji izlazi iz  $i$  s pozitivnim rezidualnim kapacitetom (u ovom slučaju brid  $(i, j)$ ), pa znamo da postoji barem jedan brid koji izlazi iz  $i$  u skupu  $S$ .

Još nam je preostalo samo vidjeti što napraviti u slučaju kada ne možemo gurnuti višak u ponor. U slučajevima kada se to dogodi, gurnut ćemo tok natrag prema izvoru: ako oznaka visine postane ili veća, onda je donja granica na udaljenost od izvora i višak u vrhu ćemo gurnuti prema izvoru.

Sada napokon možemo napisati guraj-promijeni visinu algoritam. Kažemo da je vrh **aktivan** ako ima pozitivan višak, to jest  $h[u] > 0$ . Ako u grafu postoji aktivan vrh, onda još nemamo tok, pa tražimo još dopustivih bridova. Ako postoji dopustiv brid, gurnemo što više toka možemo po njemu: gurnemo minimum između viška u vrhu i rezidualnog kapaciteta brida. Ukoliko ne postoji dopustiv brid, promijenimo visinu vrha.

**Algoritam 2:** Algoritam guraj-promijeni visinu  
**Guraj :**

**PromijeniVisinu:**

**Guraj-PromijeniVisinu:**

```
dok postoji aktivan vrh :  
  ako postoji vrh takav da je brid dopustiv:  
    Guraj  
  inače:  
    PromijeniVisinu  
  vrati
```

Prvo ćemo dokazati da [algoritam guraj-promijeni visinu](#) pronalazi maksimalan tok za svaku mrežu na ulazu. Započnimo s dokazivanjem da algoritam održava predtok i valjanu oznaku visine.

**Lema 2.4.** *Guraj-promijeni visinu algoritam održava predtok.*

*Dokaz.* Prvo moramo pokazati da je na početku algoritma stvarno predtok. Za svaki vrh takav da postoji brid na početku vrijedi  $h[u] = 0$ . Za sve ostale vrhove vrijedi  $h[u] \leq 0$ . Tada je jasno da su ograničenja kapaciteta i svojstvo iskrivljene simetrije ispoštovani, a višak je nenegativan, pa je stvarno predtok.

Pretpostavimo sada da u nekoj iteraciji algoritma imamo predtok te da smo gurnuli tok u brid  $(u, v)$ . Neka je vrijednost koju smo dobili time. Iz algoritma je vidljivo da smo gurnuli  $\min(h[u], c_{uv} - f_{uv})$  jedinica u brid  $(u, v)$ . Zaključujemo da je

ograničenju kapaciteta ispoštovano budući da vrijedi:  $\sum_{v \in V} c_v = 0$  i  $\sum_{v \in V} c_v = 0$ . Svojstvo iskrivljene simetrije je ispunjeno budući da vrijedi:  $\sum_{v \in V} c_v = 0$ . Konačno, vrhovi imaju nenegativan višak budući da smo gurnuli tok u vrhu  $v$  pa vrijedi  $c_v \geq 0$  i  $c_v = 0$ . Time smo dokazali da je predtok.  $\square$

**Lema 2.5.** *Guraj-promijeni visinu algoritam održava valjanu oznaku visine.*

*Dokaz.* Na početku algoritma postavljamo  $c_v = 0$  i  $c_v = 0$ . Jedini bridovi za koje je tada moguće da je  $c_v > 0$  su bridovi oblika  $(u, v)$  budući da je  $c_u = 0$ , a  $c_v = 0$ . Međutim, svi takvi bridovi su na početku zasićeni, pa ne može vrijediti  $c_v > 0$  budući da ti bridovi nisu u skupu  $E$ .

Sada moramo pokazati da ostaje valjana oznaka visine tijekom izvođenja algoritma. To slijedi po samoj konstrukciji operacije. Promotrimo sada što se dogodi nakon izvođenja operacije guranja. Povećavanje vrijednosti  $c_v$  može uzrokovati da rezidualni kapacitet brida  $(u, v)$  postane pozitivan. Međutim, budući da operaciju guranja izvršavamo samo na bridovima za koje vrijedi  $c_v > 0$ , znamo da vrijedi  $c_v = 0$ , pa je oznaka visine i dalje valjana. Konačno, nikad nećemo promijeniti oznaku visine vrhovima i budući da ti vrhovi nikad nisu aktivni, pa je  $c_v = 0$  i kroz cijelo izvršavanje algoritma.  $\square$

Da bi pokazali da algoritam završi s tokom  $f$ , prvo moramo dokazati iduću lemu.

**Lema 2.6.** *Za svaki predtok  $f$  i za svaki vrh  $v$  takav da vrijedi  $c_v > 0$  postoji jednostavan put od  $v$  do izvora takav da svaki brid na tom putu ima pozitivan rezidualni kapacitet.*

*Dokaz.* Neka je vrh  $v$  takav da vrijedi  $c_v > 0$  i neka je skup svih vrhova  $S$  takav da postoji put od  $v$  do koji sadrži samo bridove s pozitivnim rezidualnim kapacitetom. Pretpostavimo suprotno, to jest da  $S = \{v\}$ .

Primijetimo da je za svaki brid  $(u, v)$  takav da je  $c_u = 0$ ,  $c_v = 0$ , pa je  $c_u = 0$ . Tada po iskrivljenoj simetriji vrijedi:  $\sum_{v \in V} c_v = 0$ . Promotrimo sada  $\sum_{v \in V} c_v$ . Po definiciji imamo:

Po iskrivljenoj simetriji, elementi sume  $\sum_{v \in V} c_v$  se svi ponište pa je jednaka 0, dok su elementi sume  $\sum_{v \in V} c_v$  svi nepozitivni. Tada zaključujemo da mora vrijediti  $\sum_{v \in V} c_v = 0$ . Međutim, budući da je predtok, mora vrijediti  $\sum_{v \in V} c_v > 0$ , pa imamo da je  $\sum_{v \in V} c_v > 0$ , to jest  $\sum_{v \in V} c_v > 0$ . Tako smo došli do kontradikcije budući da je  $\sum_{v \in V} c_v = 0$ , pa početna tvrdnja vrijedi.  $\square$

Iako nam se lema 2.6 na prvi pogled čini nebitna za ovo što radimo, ona je bitna u dokazivanju da su oznake visine odozgo ograničene, što će nam biti bitno u pronalaženju složenosti algoritma.

**Lema 2.7.**  $\square$



*Dokaz.* Primijetimo da mijenjamo visinu samo vrhovima koji imaju pozitivan višak, a po lemi 2.6 znamo da uvijek postoji jednostavan put od do izvora koji sadrži samo bridove s pozitivnim rezidualnim kapacitetom. Budući da je jednostavan put, vrijedi , a jer je valjana oznaka visine  $i$  za sve  $j$  vrijedi , mora vrijediti: . □

Koristeći lemu 2.7, lako možemo ograničiti broj operacija promjene visine u algoritmu, a s malo više truda i broj operacija guranja. Krenimo s operacijama promjene visine.

**Lema 2.8.** *Broj operacija promjene visine u algoritmu guraj-promijeni visinu je .*

*Dokaz.* Sve oznake visine za vrhove na početku su postavljene na , a svaki puta kada izvršimo operaciju promjene visine, za neki vrh se njegova oznaka visine poveća barem za 1. Po lemi 2.7, svaka oznaka visine može biti najviše , a u grafu imamo vrhova kojima možemo mijenjati visinu, pa zaključujemo da ukupan broj poziva operacije promjene visine u jednom izvođenju algoritma može biti najviše . □

Da bismo mogli ograničiti broj poziva operacija guranja u algoritmu, prvo operacije guranja dijelimo u dvije vrste. Prisjetimo se da u operaciji guranja tok na bridu povećavamo za minimum između  $i$  i  $j$ . Operaciju guranja u kojoj vrijednost toka povećamo za nazivamo **zasićenom** operacijom guranja, dok onu u kojoj povećamo za nazivamo **nezasićenom** operacijom guranja. Osim toga, da bismo mogli ograničiti broj operacija guranja potrebna nam je još jedna propozicija.

**Propozicija 2.9.** *Ako je u trenutačnoj iteraciji algoritma, predtok u idućoj  $i$ , onda mora vrijediti , a time  $i$  .*

*Dokaz.* Ako je u trenutačnoj iteraciji, a u sljedećoj iteraciji je , onda je algoritam između te dvije iteracije morao povećati vrijednost toka na bridu . Iz toga zaključujemo da je , pa iz svojstva iskrivljene simetrije slijedi . □

**Lema 2.10.** *Broj zasićenih operacija guranja u algoritmu guraj-promijeni visinu je .*

*Dokaz.* Neka je brid proizvoljan. Ako smo izvršili zasićenu operaciju guranja na bridu , onda je taj brid bio dopustiv i . Budući da brid tada postane zasićen, njegov rezidualni kapacitet je 0. Iz propozicije 2.9 zaključujemo da tada brid može imati rezidualni kapacitet veći od 0 samo ako se u nekoj iteraciji poveća tok na bridu , što se dogodi ako izvršimo operaciju guranja na bridu .

Neka je oznaka visine u iteraciji u kojoj je pušten tok u bridu  $e$ . Iz toga slijedi da je u toj iteraciji moralo vrijediti  $h(e) \geq 2$ . Jer su oznake visine nepadajuće tijekom izvođenja algoritma, vrijedi  $h(e) \geq 2$ . To znači da se oznaka visine vrha mora povećati barem za 2 prije nego što brid  $e$  ponovno može imati pozitivan rezidualni kapacitet, a time je moguće i izvršiti novu zasićenu operaciju guranja nad bridom  $e$ .

Budući da po lemi 2.7 vrijedi  $h(e) \geq 2$ , zaključujemo da možemo izvršiti najviše zasićenih operacija guranja na bridu  $e$  tijekom izvođenja algoritma. Budući da imamo ukupno bridova u grafu, broj zasićenih operacija guranja tijekom izvođenja algoritma je najviše  $\sum_{e \in E} h(e)$ .  $\square$

Ako zamislimo da oznaka visine vrha označava na kojoj se visini vrh nalazi (što sami naziv "oznaka visine" sugerira), onda dokaz prijašnje leme sugerira da zasićena operacija guranja gura tok nizvodno, te da bismo mogli gurnuti tok iz  $u$  u  $v$  ponovno, prije toga ga moramo gurnuti iz  $u$  u  $v$ . Da bismo mogli gurnuti tok nizvodno, moramo povećati visinu čvora barem za 2, a budući da visina vrha može biti najviše  $h$ , to znači da možemo izvršiti najviše zasićenih operacija guranja na bridu  $e$ .

Sada ćemo ograničiti broj nezasićenih operacija guranja tijekom izvršavanja algoritma. Da bismo to napravili, koristit ćemo **argument potencijalne funkcije**. Definirat ćemo potencijalnu funkciju koja će kao argumente primati neke parametre algoritma i vrijednost te funkcije će biti nenegativna tijekom izvođenja algoritma. Na početku algoritma, vrijednost funkcije biti će neki nenegativan broj  $\Phi$ . Svaki korak algoritma koji će nam biti interesantan (primjerice svako izvođenje nezasićene operacije guranja) uzrokovat će smanjenje vrijednosti funkcije za barem 1. Ostali koraci, nama nezanimljivi, mogu uzrokovati povećanje vrijednosti funkcije  $\Phi$ . Da bismo ograničili broj nama zanimljivih koraka, dovoljno je ograničiti za koliko se vrijednosti funkcije poveća tijekom izvođenja algoritma. Naime, broj izvođenja nama zanimljivih koraka može biti najviše plus suma svih povećanja vrijednosti funkcije  $\Phi$ .

**Lema 2.11.** *Broj nezasićenih operacija guranja u algoritmu guraj-promijeni visinu je  $\Phi$ .*

*Dokaz.* Neka je  $\Phi$  početna vrijednost funkcije. Tada je  $\Phi$  na početku i na kraju algoritma jer na početku su oznake visine svih aktivnih vrhova jednake 0, a na kraju algoritma nijedan vrh nije aktivan. Promotrimo sada koje operacije uzrokuju promjene u vrijednosti funkcije  $\Phi$ .

Vrijednost funkcije se smanjuje nakon svakog izvođenja nezasićene operacije guranja, budući da izvođenje te operacije pretvori jedan aktivan vrh u neaktivan i time ga miče iz sume. Da bismo izvršili operaciju guranja na bridu  $e$ , treba vrijediti  $h(e) \geq 2$ , a čak i ako nezasićena operacija guranja pretvori vrh u aktivan vrh, ukupna promjena vrijednosti funkcije je  $-1$ .

Vrijednost funkcije se povećava nakon izvođenja operacije promjene visine i zasićene operacije guranja koja stvara novi aktivan vrh. Operacija promjene visine može povećati vrijednost funkcije tako što će povećati oznaku visine, a dodavanje novog aktivnog vrha u sumu može povećati vrijednosti funkcije za  $h(v)$ . Tada se može ukupno povećati za najviše zbog operacija promjene visine, a  $h(v)$  zbog zasićenih operacija guranja.

Budući da je na početku izvođenja algoritma  $\Phi$ , ukupan broj nezasićenih operacija guranja tijekom izvođenja algoritma je  $\Phi$ .  $\square$

Sada možemo sve ove rezultate spojiti u jedan i dobiti složenost algoritma guraj-promijeni visinu.

**Teorem 2.12.** *Algoritam guraj-promijeni visinu izračuna maksimalan tok u koraka.*

*Dokaz.* Svaka operacija guranja je složenosti  $O(E)$ , pa je ukupna složenost svih operacija guranja u algoritmu  $O(V \cdot E)$ . Za svaki vrh čuvamo listu bridova koji izlaze iz njega sortiranu po vremenku kada smo zadnji put gurnuli tok po tom bridu. Kada tražimo dopustiv brid koji izlazi iz  $v$ , krenemo od posljednjeg brida na kojem je bio gurnut tok; ako taj brid nije dopustiv, prijeđemo na sljedeći. Ako smo došli do kraja liste, ne postoji dopustiv brid iz vrha  $v$ , pa možemo izvršiti operaciju promjene visine. Provjeravanje svih bridova koji izlaze iz  $v$  se može napraviti u  $O(E)$  koraka. Po lemi 2.7, operaciju promjene visine napravimo puta  $O(V)$ , tako da je ukupno vrijeme izvođenja svih operacija promjene visine  $O(V \cdot E)$ .

Kada izvršavanje algoritma stane, je predtok  $f$ . Tada po lemi 2.3 ne postoji povećavajući put u grafu  $G$ , pa je maksimalan tok  $f$ .  $\square$

### 3 Implementacija algoritma guraj-promijeni visinu

Kao što smo spomenuli u uvodu, za implementiranje svih verzija guraj-promijeni visinu algoritma koristit ćemo programski jezik Javascript, točnije biblioteku React. U ovoj sekciji opisat ćemo implementaciju [algoritma guraj-promijeni visinu](#) i struktura koje su nam potrebne. Implementacija algoritma se može naći u [GitHub repozitoriju \[3\]](#).

Osnovna struktura koja nam je potrebna u implementaciji je sam graf. Graf je objekt koji sadrži dva niza podataka: niz vrhova i niz bridova. Svaki vrh je objekt koji sadrži informaciju o vrijednosti oznake visine tog vrha i vrijednosti viška toka u tom vrhu. Svaki brid je objekt koji sadrži oznaku početnog i krajnjeg vrha tog brida, kapacitet i tok. Na početku, niz bridova je prazan, dok niz vrhova sadrži 2 vrha (tako osiguravamo da svaki graf ima izvor i ponor). Početnu vrijednost grafa možemo vidjeti u idućem isječku koda.

```
1 graph: {
2   nodes: [{
3     height: 0,
4     excess: 0
5   }, {
6     height: 0,
7     excess: 0
8   }],
9   edges: []
10 }
```

### Kod 1: Osnovni graf prije unošenja podataka

Promotrimo sada implementaciju dviju potrebnih operacija: operacije guranja i operacije promjene visine. Operacija guranja danom grafu poveća tok za minimum između vrijednosti viška na početnom vrhu i rezidualnog kapaciteta danog brida. Operacija promjene visine postavi vrijednost oznake visine danog vrha na sljedbenika minimuma visina svih susjednih vrhova koji su spojeni s danim vrhom nezasićenim bridom.

```
1  const push = (graph, startNodeIndex, admissibleEdgeIndex) => {
2    let delta = Math.min(graph.nodes[startNodeIndex].excess,
3      graph.edges[admissibleEdgeIndex].capacity -
4      graph.edges[admissibleEdgeIndex].flow);
5    graph.edges[admissibleEdgeIndex].flow += delta;
6    graph.edges[admissibleEdgeIndex +
7      (admissibleEdgeIndex % 2 === 0 ? 1 : -1)].flow -= delta;
8    setExcess(graph.nodes, graph.edges);
9  };
```

## Kod 2: Operacija guranja

```
1  const relabel = (graph, i) => {
2    let positiveResidualEdges = graph.edges.filter(curr =>
3      curr.startNode === i + 1 && curr.flow !== curr.capacity);
4    let neighbourNodesIndexes = positiveResidualEdges.map(elem =>
5      elem.startNode === i + 1 ? elem.endNode : elem.startNode);
6    let neighbourNodes = neighbourNodesIndexes.map(elem =>
7      graph.nodes[elem - 1].height);
8    graph.nodes[i].height = Math.min(...neighbourNodes) + 1;
9  };
```

### Kod 3: Operacija promjene visine

Jednom kada imamo operacije guranja i promjene visine možemo implementirati guraj-promijeni visinu algoritam.

```
1  const pushRelabel = (graph) => {
2    graph.edges = setInitialFlow(graph.edges);
3    graph.nodes = setInitialHeight(graph.nodes);
4
5    graph.nodes = setExcess(graph.nodes, graph.edges);
6
7    while( help.excessSum(graph.nodes) !== 0 ) {
8      let startNodeIndex = findFirstActive(graph.nodes);
9      let admissibleEdgeIndex = admissibleEdge(startNodeIndex, graph.edges, graph.nodes);
10     if(startNodeIndex !== -1 && admissibleEdgeIndex !== -1) {
11       push(graph, startNodeIndex, admissibleEdgeIndex);
12     } else {
13       relabel(graph, startNodeIndex);
14     }
15   }
16
17   return graph;
18 };
```

#### Kod 4: Algoritam guraj-promijeni visinu

U implementiranim algoritmima koje smo pokazali možemo vidjeti da se koriste dodatne metode. Dvije metode služe nam za postavljanje početnih vrijednosti za neke vrhove i bridove: `setInitialFlow` i `setInitialHeight`. Metoda `setInitialFlow` postavlja vrijednost toka svih bridova koji izlaze iz izvora na kapacitet tog brida, a vrijednost toka svih bridova koji ulaze u izvor na minus kapacitet tog brida. Metoda `setInitialHeight` postavlja visinu izvora na broj vrhova u grafu.

Preostale dodatne metode korištene u implementaciji algoritma su `excessSum`, `findFirstActive` i `admissibleEdge`. Metoda `excessSum` računa sumu viška svih vrhova osim izvora i ponora (u implementaciji se spominje skaliranje u valovima, to će nam biti bitno za jednu varijantu ovog algoritma). Metoda `findFirstActive` pronalazi prvi aktivan vrh u grafu, a metoda `admissibleEdge` indeks prvog valjanog brida u grafu.

Metoda `setExcess` koristi se u implementaciji operacije guranja, prolazi po svim vrhovima grafa te za svakoga računa višak. Implementacije svih ovih metoda možete vidjeti u nastavku.

```
1  const setInitialFlow = (edges) => {
2    for(let i = 0; i < edges.length; i += 2) {
3      if(edges[i].startNode === 1) {
4        edges[i].flow = edges[i].capacity;
5        edges[i + 1].flow = (-1) * edges[i].capacity;
6      }
7    }
8    return edges;
9  };
10
11 const setInitialHeight = (nodes) => {
```

```

12     nodes[0].height = nodes.length;
13     return nodes;
14 }
15
16 const excessSum = (nodes, isWaveScaling) => nodes.reduce((partialSum, elem) => partialSum + ( isWaveScaling &&
(elem.label == nodes.length - 1) ? 0 : elem.excess ), -nodes[0].excess);
17
18 const findFirstActive = (nodes) => nodes.findIndex(elem => isNodeActive(nodes.length, elem));
19
20 const isNodeActive = (numOfNodes, node) => node.label !== 0 &&
21     node.label !== numOfNodes - 1 && node.excess > 0;
22
23 const admissibleEdge = (i, edges, nodes, isExcessScaling=false, deltaScale=0) => {
24     for(let edgeInd in edges) {
25         if(edges[edgeInd].startNode === i + 1
26             && nodes[edges[edgeInd].endNode - 1].height
27                 === nodes[i].height - 1
28             && edges[edgeInd].flow !== edges[edgeInd].capacity
29             && ( isExcessScaling
30                 ? deltaScale > nodes[edges[edgeInd].endNode - 1].excess
31                 : true ) ) {
32             return parseInt(edgeInd);
33         }
34     }
35     return -1;
36 };
37
38 const setExcess = (nodes, edges) => {
39     for(let i = 1; i < nodes.length; i++) {
40         nodes[i - 1].excess = 0;
41         for(let edge of edges) {
42             if(edge.endNode === i) {
43                 nodes[i - 1].excess += edge.flow;
44             }
45         }
46     }
47     return nodes;
48 };

```



## Kod 5: Pomoćne metode

Još je samo preostalo komentirati činjenicu da nam u implementaciji algoritam vraća cijeli graf, a ne maksimalan tok. To je napravljeno da bi se omogućio grafički prikaz toka, a iz grafa računamo tok tako da zbrojimo vrijednosti toka u bridovima koji izlaze iz izvora.

```
1  const calculateMaxFlow = () =>
2    pushRelabel(structuredClone(state.graphForAlgs)).edges
3    .reduce((partialSum, elem) => partialSum +
4      (elem.startNode === 1 && elem.endNode !== 1 ? elem.flow : 0)
5      , 0);
```

## Kod 6: Metoda za računanje maksimalnog toka

## 4 Varijante algoritma guraj-promijeni visinu

Kod analiziranja osnovne verzije algoritma guraj-promijeni visinu može se vidjeti da je vrijeme izvođenja dominirano operacijama guranja. Guraj-promijeni visinu algoritam vrlo je fleksibilan: imamo dosta slobode u načinu biranja bridova po kojima ćemo gurati tok i vrhova kojima ćemo promijeniti visinu. Iskorištavanjem ova fleksibilnosti možemo ubrzati osnovnu verziju algoritma. Iz teorema 2.12 očito je da je složenost guraj-promijeni visinu algoritma zbog nezasićenih operacija guranja. Da bismo smanjili vremensku složenost tog algoritma, uvodimo operaciju **ispražnjavanja**. Operacija ispražnjavanja uzima neki aktivan vrh te izvršava operacije guranja i promjene visine sve dok taj vrh ne postane neaktivan. Primijetimo da se u svakom izvođenju operacije ispražnjavanja nezasićena operacija guranja izvrši samo jednom, jer samo zadnja operacija guranja na aktivnom vrhu može biti nezasićena. U nastavku možete vidjeti implementaciju operacije ispražnjavanja.

```
1  const discharge = (graph, activeNodes, i) => {
2    while( graph.nodes[i].excess > 0 ) {
3      let nextEdge = admissibleEdge(i, graph.edges, graph.nodes);
4      while( nextEdge !== -1 && graph.nodes[i].excess !== 0 ) {
5        pushWithDischarge(graph, i, nextEdge, activeNodes);
6        nextEdge = admissibleEdge(i, graph.edges, graph.nodes);
7      }
8      if( graph.nodes[i].excess > 0 ) {
9        relabel(graph, i);
10     }
11   }
12 };
```

## Kod 7: Implementacija operacije ispražnjavanja

Možete vidjeti da se kod implementacije operacije ispražnjavanja ne koristi klasična varijanta operacije guranja, već varijanta operacije guranja kod ispažnjavanja. Naime, kod nekih varijanti algoritma moramo održavati red aktivnih vrhova, pa kod svake operacije guranja moramo provjeriti je li možda krajnji vrh brida po kojem smo gurali tok postao aktivan. Implementaciju nove varijante operacije guranja možete vidjeti u nastavku.

```
1  const pushWithDischarge = (graph, startNodeIndex, admissibleEdgeIndex, activeNodes) => {
2    let delta = Math.min(graph.nodes[startNodeIndex].excess,
3      graph.edges[admissibleEdgeIndex].capacity
4      - graph.edges[admissibleEdgeIndex].flow);
5    graph.edges[admissibleEdgeIndex].flow += delta;
6    graph.edges[admissibleEdgeIndex + (admissibleEdgeIndex % 2 === 0 ? 1 : -1)].flow -= delta;
7    setExcess(graph.nodes, graph.edges);
8
9    let endNode = graph.nodes
10     [graph.edges[admissibleEdgeIndex].endNode - 1];
11    if(isNodeActive(graph.nodes.length, endNode)
12      && (!activeNodes.includes(endNode))) {
13      activeNodes.push(endNode);
14    }
15  };
```

## Kod 8: Implementacija operacije ispražnjavanja

### 4.1 FIFO guraj-promijeni visinu algoritam

Prva varijanta algoritma guraj-promijeni visinu koju ćemo promatrati zove se **FIFO guraj-promijeni visinu algoritam**. Ova varijanta održava red aktivnih vrhova: na početku algoritma svi su aktivni vrhovi ubačeni u red. Algoritam uzima vrh s početka reda i izvrši operaciju ispražnjavanja nad njim. Ukoliko neki drugi vrh postane aktivan tijekom izvršavanja te operacije, taj vrh se stavlja na kraj reda.

Da bismo odredili vremensku složenost ove varijante guraj-promijeni visinu algoritma, sve što trebamo je odozgo omeđiti broj nezasićenih operacija guranja izvršenih tijekom izvršavanja algoritma. Za to koristimo argument potencijalne funkcije na prolascima po redu. Prvi prolaz po redu završava kada je algoritam izvršio operaciju ispražnjavanja nad svim vrhovima koji su stavljeni u red na početku izvršavanja algoritma. Općenito,  $i$ -ti prolaz po redu završava kada je algoritam izvršio operaciju ispražnjavanja nad svim vrhovima koji su dodani u red u  $i$ -om prolazu po redu.

U razmatranju vremenske složenosti koristi se potencijalna funkcija  $f(i)$ . Korištenjem te potencijalne funkcije, nije teško vidjeti da se tijekom izvršavanja algoritma odradi prolaza po redu, a to povlači da je broj izvršavanja nezasićenih operacija guranja u ovoj verziji algoritma  $O(n)$ . Iz toga slijedi da je vremenska složenost FIFO guraj-promijeni visinu algoritma  $O(n)$ .

## 4.2 Guraj-promijeni visinu algoritam sa skaliranjem viška

Iduća varijanta algoritma guraj-promijeni visinu poznata je pod imenom **guraj-promijeni visinu algoritam sa skaliranjem viška**. U ovoj varijanti kroz izvođenje algoritma održavamo parametar kojeg na početku izvođenja postavljamo na  $h$ . Algoritam se izvršava kao niz **faza -skaliranja**. Parametar koristimo u ograničavanju promjene vrijednosti predtoka, točnije održavamo ga  $h$ -optimalnim.

**Definicija 4.1.** Za predtok kažemo da je **-optimalan** ako vrijedi  $h$ .

U svakoj fazi  $h$ -skaliranja koristimo modificirane verzije operacije guranja i promjene visine da bi predtok ostao  $h$ -optimalan. Korisno će nam biti definirati još jedan pojam.

**Definicija 4.2.** Za vrh kažemo da je **-aktivan** ako vrijedi  $h$ .

U ovoj varijanti algoritma, u svakoj fazi  $h$ -skaliranja, odaberemo  $h$ -aktivan vrh s najmanjom visinom te tada gurnemo višak po dopustivim bridovima (prisjetimo se da je brid dopustiv ako ima pozitivan rezidualni kapacitet i ako je  $h$ ). Nastavljamo gurati po dopustivim bridovima koji izlaze iz vrha sve dok višak ne postane manji ili jednak  $h$  ili dok ne moramo vrhu promijeniti visinu. Tada odaberemo idući  $h$ -aktivan vrh s najmanjom visinom. Faza  $h$ -skaliranja završava kada više ne postoji nijedan  $h$ -aktivan vrh. Tada podijelimo s 2 i prijedemo na iduću fazu  $h$ -skaliranja. Primijetimo da, jer na kraju neke faze više nema  $h$ -aktivnih vrhova, predtok je  $h$ -optimalan i za novu vrijednost  $h/2$ .

Može se vidjeti da ovako definiran algoritam staje za  $h$  te da je u toj iteraciji pronađen maksimalan tok. Također, korištenjem potencijalne funkcije  $h$  može se dokazati da je broj nezasićenih operacija guranja u svakoj fazi  $h$ -skaliranja odozgo ograničen s  $h$ , pa iz toga vidimo da je vremenska složenost guraj-promijeni visinu algoritma sa skaliranjem viška  $O(n \log V)$ .

## 4.3 Guraj-promijeni visinu algoritam sa skaliranjem u valovima

Sljedeća varijanta algoritma guraj-promijeni visinu koju ćemo opisati je **Guraj-promijeni visinu algoritam sa skaliranjem u valovima**. Da bismo mogli implementirati ovu varijantu algoritma, uvodimo novu verziju operacije guranja, takozvana **operacija guranja sa stogom**. Za svaki vrh  $v$ , čuvamo nekako sortiran niz svih bridova koji izlaze iz  $v$ . Tijekom izvođenja operacije guranja sa stogom, prolazimo po nizu bridova koji izlaze iz  $v$  i pokušamo gurnuti tok na svakom. Nakon što smo prošli sve bridove koji izlaze iz  $v$ , znamo da je vrijeme da promijenimo visinu vrha  $v$ .

**Algoritam 3:** Operacija guranja sa stogom**GurajSaStogom** za :

dodaj u prazan stog

**dok** je stog neprazan:

neka je vrh na vrhu stoga

neka je trenutni brid čiji je početni vrh

**ako** nije dopustiv:**ako** je zadnji brid iz vrha :

postavi da je prvi brid iz trenutni brid

izbaci iz

PromijeniVisinu

**inače:**

postavi da je idući brid iz trenutni brid

**inače ako** je i :

gurni na stig

**inače:**

Guraj

**ako** je :

izbaci sa stoga

Guraj-promijeni visinu algoritam sa skaliranjem u valovima izvršava se u nizu **valova**. U svakom valu, prvo sortiramo sve vrhove u nerastućem redoslijedu po oznaci visine. Nakon toga prolazimo po sortiranom nizu vrhova te ako je vrh aktivan i još mu u ovom valu nismo promijenili visinu, izvršimo operaciju guranja sa stogom na vrhu .

Definiramo ukupni višak kao sumu viška svih vrhova , osim izvora i ponora, točnije . Neka je parametar čiju ćemo vrijednost definirati kasnije. Algoritam će izvoditi niz valova sve dok je ukupni višak veći ili jednak od . Onda će se ponašati kao varijanta sa skaliranjem viška i izvršit će operaciju guranja sa stogom nad svim vrhovima za koje vrijedi . To će raditi sve dok postoji vrh čiji je višak veći od , a zatim je trenutna faza - skaliranja gotova i tada podijelimo s 2. Cijeli algoritam možete vidjeti u nastavku.

**Algoritam 4:** Guraj-promijeni visinu algoritam sa skaliranjem u valovima**dok** je :**dok** je :

neka je vrhova silazno sortiran po

**za** svaki :**ako** je i vrhu još nije promijenjena visina u ovom valu:

GurajSaStogom

**dok** postoji vrh takav da je :

vrati

Kod promatranja vremenske složenosti algoritma, podijelit ćemo operacije guranja u dvije vrste: svaku operaciju guranja koja gurne barem  $\epsilon$  jedinica toka nazivamo **velikom** operacijom guranja, a one koje gurnu manje od  $\epsilon$  jedinica toka **malom** operacijom guranja.

Može se vidjeti da, svaki put kada je neki vrh na stogu tijekom izvršavanja operacije guranja sa stogom, se za taj vrh izvrše najviše dvije nezasićene operacije guranja, takve da je najviše jedna mala. Također, na kraju vala, svakom bridu koji ima pozitivan rezidualni kapacitet je bila promijenjena visina. Koristeći te dvije tvrdnje i potencijalnu funkciju kao u promatranju složenosti varijante s skaliranjem viška, može se dokazati da se tijekom izvršavanja algoritma izvrši najviše  $\epsilon$  nezasićenih velikih operacija guranja i toliko nezasićenih malih operacija guranja plus nezasićenih malih operacija guranja po valu.

Može se također vidjeti da, u svim fazama -skaliranja osim u zadnjoj, se u svakom valu izvrši operacija promjene visine. Ako se u valu nije izvršila nijedna operacija promjene visine, tada nema vrhova s pozitivnim viškom na kraju vala, pa algoritam staje.

Iz svih ovih tvrdnji može se zaključiti da se tijekom izvršavanja algoritma izvrši  $\epsilon$  operacija guranja. Tada, uz odabir  $\epsilon$  dobivamo da je složenost guraj-promijeni visinu algoritam sa skaliranjem u valovima  $O(n \cdot \epsilon)$ .

**Napomena:** Članak je nastao iz diplomskog rada studenta Maria Zečića, pod mentorstvom izv. prof. dr. sc. Ivice Nakića, na Diplomskom sveučilišnom studiju Računarstvo i matematika Matematičkog odsjeka Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu.

## Bibliografija

- [1] David P. Williamson. *Network Flow Algorithms*. Cambridge University Press, 2019.
- [2] Članak na Wikipediji o Guraj-Promijeni visinu algoritmu [Push-relabel maximum flow algorithm](#).
- [3] GitHub repozitorij implementacije <https://github.com/Mario173/PushRelabel>.



