Algoritmi za problem toka kroz mrežu

Surić, Josipa

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Josipa Surić

ALGORITMI ZA PROBLEM TOKA KROZ MREŽU

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, rujan, 2018

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Sadržaj **iii**

Uvod **1**

1 Tok kroz mrežu **2**
1.1 Definicije 2

2 Ford-Fulkersonov algoritam **5**
2.1 Definicije 5
2.2 Opis algoritma 8
2.3 Minimaln rez 10
2.4 Teorem maksimalnog toka 10
2.5 Složenost Ford-Fulkersonovog algoritma 12
2.6 Primjer Ford-Fulkersonovog algoritma 12

3 Edmonds-Karpov algoritam **15**
3.1 Opis algoritma 15
3.2 Složenost Edmonds-Karpovog algoritma 15
3.3 Primjer Edmonds-Karpovog algoritma 16
3.4 Edmonds-Karpov algoritam na računalu 18

4 Proširene verzije problema toka kroz mrežu **20**
4.1 Prvo proširenje: Cirkulacije sa zahtjevima 20
4.2 Drugo proširenje: Cirkulacije sa zahtjevima i donjim granicama 21

5 Primjene problema toka kroz mrežu **23**
5.1 Provođenje nadzora 23
5.2 Raspored letenja 26

Bibliografija **30**

Uvod

Problem toka kroz mrežu predstavlja važnu skupinu optimizacijskih problema te je jedan od ključnih problema u operacijskim istraživanjima, računarstvu i kombinatorici. U ovom radu prvenstveno ćemo se posvetiti problemu pronalaska maksimalnog toka kroz mrežu. Problem pronalaska maksimalnog toka kroz mrežu predmet je proučavanja još od 1950-ih. Ima mnoge primjene u transportu, prijevozu, telekomunikacijama i izradi rasporeda. Mi ćemo navesti samo neke od tih primjena. Postoje mnogi efikasni algoritmi za rješavanje ovog problema, poput Ford-Fulkersonovog algoritma i Edmonds-Karpovog algoritma, koje ćemo opisati u poglavljima 2. i 3. Prije samog opisa navedenih algoritama, donosimo neke osnovne definicije za bolje razumijevanje problema. Nakon opisa algoritama, navodimo proširenja problema toka kroz mrežu, koja su nam potrebna za postavljanje i rješavanje problema provođenja nadzora i određivanja rasporeda letenja.

Poglavlje 1

Tok kroz mrežu

1.1 Definicije

Kao što smo već naveli, problem toka kroz mrežu označava skupinu važnih optimizacijskih problema koji se mogu prikazati pomoću usmjerenih grafova. Definiramo pojam mreže (eng. flow network) kao usmjereni graf s nenegativnim težinama bridova. Bridove grafa slikovito zamišljamo kao cijevi koje prenose, primjerice, megabajte ili neku tekućinu, ovisno o problemu kojeg rješavamo. Svaki brid grafa ima zadan kapacitet koji ograničava količinu tvari koje prenosimo kroz mrežu. Osim kapaciteta, graf ima posebno naznačen početni vrh, kojeg nazivamo izvor, te krajnji vrh, kojeg nazivamo ponor. Ideja je pronaći najveći ukupni protok tvari od izvora do ponora.

Sljedeće definicije i primjeri su preuzeti iz [5].

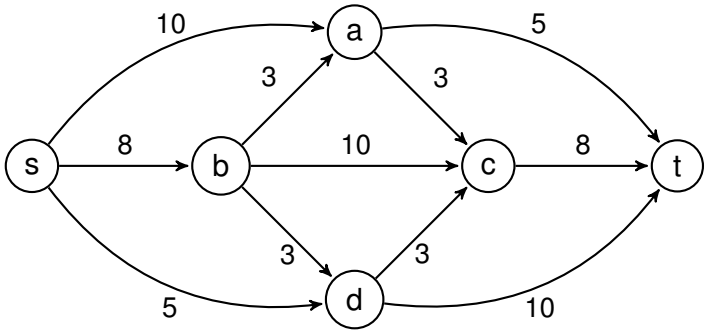
Definicija 1.1.1. *Mreža je usmjereni graf $G = (V, E)$ u kojem svaki brid $(u, v) \in E$ ima nenegativan kapacitet $c(u, v) \geq 0$. Za $(u, v) \notin E$ definiramo $c(u, v) = 0$. Mreža sadrži dva posebno naznačena vrha: izvor s i ponor t.*

Pretpostavljamo da ne postoji brid koji ulazi u s, kao ni brid koji izlazi iz t. Takvu mrežu ponekad nazivamo *s-t* mreža. Osim toga, pretpostavljamo da se svaki vrh grafa nalazi na nekom putu od izvora do ponora. Iz toga slijedi $m \geq n - 1$, pri čemu je $n = |V|$ i $m = |E|$. Također, pretpostavljamo da su vrijednosti kapaciteta cijeli brojevi.

Primjer 1.1.2. *Jedan primjer mreže prikazan je na slici 1.1.*

Definicija 1.1.3. *Za danu s-t mrežu, tok (ili s-t tok) definiramo kao funkciju f koja preslikava svaki brid u nenegativan realan broj i zadovoljava sljedeće uvjete:*

1. *Ograničenje kapaciteta: Za svaki brid $(u, v) \in E$, vrijedi $f(u, v) \leq c(u, v)$.*



Slika 1.1: Mreža

2. *Očuvanje toka: Za sve* $v \in V \setminus \{s, t\}$*, ukupan tok kroz bridove koji ulaze u vrh* v*, jednak je sumi toka kroz bridove koji izlaze iz vrha* v*.*

Zahtjev očuvanja toka možemo i formalnije zapisati. Pretpostavimo najprije da za svaki brid (u, v), koji nije iz E, vrijedi $f(u, v) = 0$. Tada ukupan tok u vrh v definiramo kao $f^{in}(v) = \sum_{(u,v)\in V} f(u, v)$, dok ukupan tok iz vrha v definiramo kao $f^{out}(v) = \sum_{(v,w)\in V} f(v, w)$. Iz zahtjeva očuvanja toka slijedi $f^{in}(v) = f^{out}(v)$, za svaki vrh $v \in V \setminus \{s, t\}$. Vrijednost $f(u, v)$ nazivamo tok kroz brid (u, v).

Definicija 1.1.4. *Ukupan tok, u oznaci* $|f|$*, definiramo kao sumu svih tokova koji izlaze iz vrha* s*.*

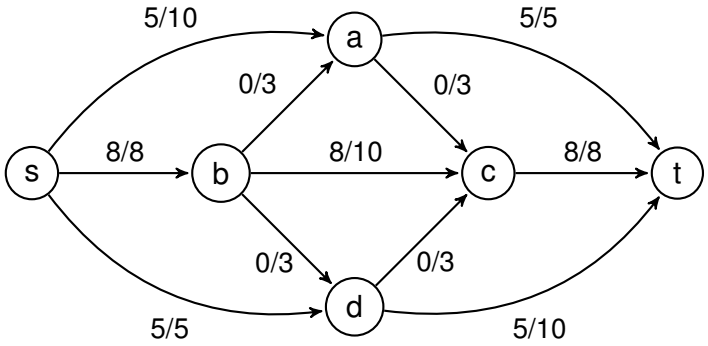
$$|f| = f^{out}(s) = \sum_{w\in V} f(s, w).$$

Definicija 1.1.5. *Problem maksimalnog toka (eng. maximum-flow problem) definiramo kao problem pronalaska toka maksimalne vrijednosti od vrha* s *do vrha* t*, za zadanu mrežu* $G = (V, E)$ *te izvor* s *i ponor* t*.*

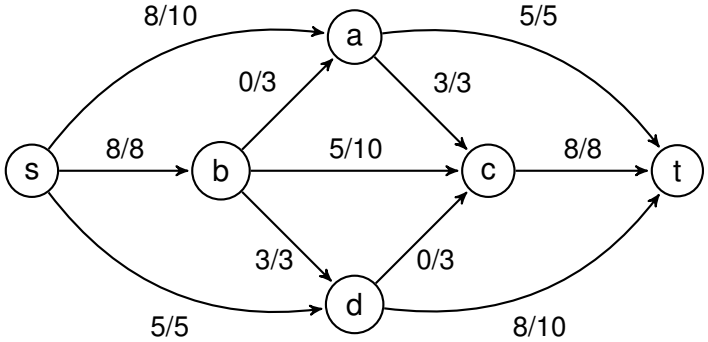
Primjer 1.1.6. *Na slikama 1.2 i 1.3 koristimo* f/c *notaciju na svakom bridu, kako bismo naznačili tok* f *i kapacitet* c*. Vrijednost toka prikazanog na slici 1.2 iznosi* $5 + 5 + 8 = 18$*, dok vrijednost maksimalnog toka prikazanog na slici 1.3 iznosi* $8 + 8 + 5 = 21$*.*

Definiciju 1.1.4 ponekad nazivamo definicija toka zasnovana na bridovima. Postoji i njoj ekvivalentna definicija, koju nazivamo definicija toka zasnovana na putu (eng. path-based flow). *s-t* put definiramo kao bilo koji put od vrha s do vrha t u grafu.

Definicija 1.1.7. *Tok zasnovan na putu je funkcija koja svakom s-t putu dodjeljuje nenegativan realan broj takav da, za svaki brid* $(u, v) \in E$*, ukupan tok kroz sve puteve koji sadrže brid* (u, v) *nije veći od* $c(u, v)$*.*



Slika 1.2: Tok kroz mrežu



Slika 1.3: Maksimalan tok kroz mrežu

Vrijednost toka zasnovanog na putu definiramo kao ukupnu sumu svih tokova na svim *s-t* putevima. Iz navedenih definicija proizlazi sljedeća tvrdnja, koju nećemo dokazivati.

Teorem 1.1.8. *Za zadanu s-t mrežu G, pod pretpostavkom da ne postoje bridovi koji ulaze u s, kao ni bridovi koji izlaze iz t, vrijednost toka zasnovanog na bridovima mreže G iznosi x, ako i samo ako mreža G ima tok zasnovan na putu koji, također, ima vrijednost x.*

Poglavlje 2

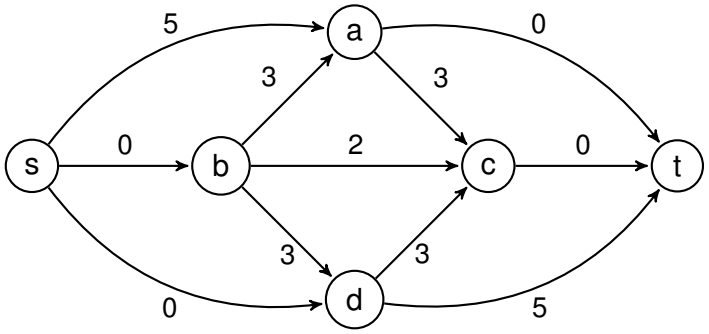
Ford-Fulkersonov algoritam

2.1 Definicije

Ford-Fulkersonov algoritam je jedan oblik pohlepnog algoritma koji pronalazi maksimalan tok u mreži. Češće ga zovemo metodom, obzirom da pristup pronalaska bridova u kojima povećavamo tok u rezidualnom grafu nije u potpunosti određen. Algoritam su objavili Lester Randolph Ford jr. i Delbert Ray Fulkerson, 1956. godine, te je jedan od najpoznatijih algoritama za rješavanje problema toka kroz mrežu. Prije nego opišemo sam algoritam, razmotrimo zašto običan pohlepni algoritam za računanje maksimalnog toka nije primjeren za rješavanje problema toka kroz mrežu. Ideja za pohlepni algoritam motivirana je pojmom toka kroz mrežu, zasnovanog na putu kroz graf. Inicijalno, tok kroz svaki brid je jednak nuli. Zatim, tražimo bilo koji put P od s do t, takav da je kapacitet svakog brida na putu P strogo pozitivan. Neka je c_{min} najmanji kapacitet bilo kojeg brida na putu P. Tu veličinu nazivamo usko grlo kapaciteta puta P. Pustimo c_{min} jedinica toka kroz put P. Za svaki brid $(u, v) \in P$, postavimo $f(u, v) \leftarrow c_{min} + f(u, v)$ i smanjimo kapacitet brida (u, v) za c_{min}. Ponavljamo postupak dok ne iskoristimo sve s-t puteve u mreži sa strogo pozitivnim kapacitetima bridova. Iako navedeni algoritam računa tok kroz mrežu, lako može zapeti kod računanja maksimalnog toka kroz mrežu. Promotrimo primjer.

Primjer 2.1.1. *Pretpostavimo da kroz mrežu, prikazanu na slici 1.1, pustimo* 5 *jedinica toka kroz najgornji put,* 8 *jedinica toka kroz srednji put te* 5 *jedinica toka kroz najdonji put. Dobili smo tok u vrijednosti* 18 *jedinica. No, nakon prilagođavanja kapaciteta, dobijemo mrežu prikazanu na slici 2.1. Uočimo da na novoj mreži koju smo dobili, nema više puteva od s do t sa strogo pozitivnim kapacitetima bridova. Tu pohlepni algoritam staje.*

Problem je u tome što ne postoji s-t put na kojem možemo direktno pustiti tok, bez prekoračenja nekog kapaciteta. Kako bismo riješili problem kojeg imamo kod pohlepnog algoritma, ideja je, pored povećanja toka u bridovima, smanjiti tok u bridovima koji već



Slika 2.1: Pohlepni algoritam

prenose tok, sve dok je tok uvijek nenegativan. Kako bismo detaljnije pojasnili ovu ideju, najprije moramo definirati rezidualnu mrežu i tzv. povećavajuće puteve (eng. augmenting paths), na kojima povećavamo tok.

Primjer 2.1.1, definicije i teoremi koje navodimo u nastavku, te pseudokod algoritma preuzeti su iz [5].

Definicija 2.1.2. *Za mrežu G i tok f, definiramo rezidualnu mrežu G_f, kao mrežu s istim skupom vrhova kao u G te istim izvorom i ponorom, čiji su bridovi definirani na sljedeći način:*

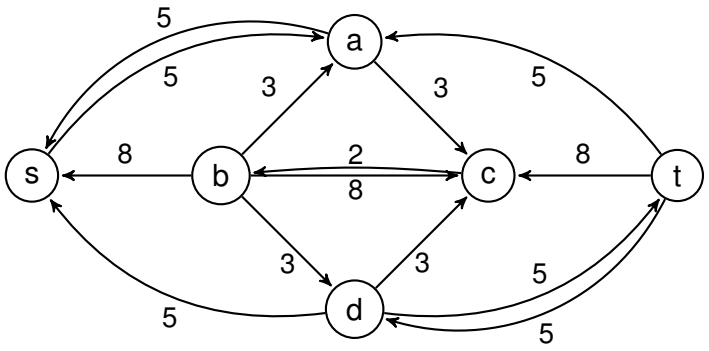
1. *Bridovi unaprijed (eng. forward edges): Za svaki brid (u, v), za koji je $f(u, v) < c(u, v)$, kreiramo brid (u, v) u mreži G_f te mu dodjeljujemo kapacitet $c_f(u, v) = c(u, v) - f(u, v)$. Ovaj brid označava da možemo dodati do $c_f(u, v)$ dodatnih jedinica toka ovom bridu, bez da narušimo izvorni zahtjev za očuvanjem kapaciteta.*

2. *Bridovi unatrag (eng. backward edges): Za svaki brid (u, v), za koji je $f(u, v) > 0$, kreiramo brid (v, u) u mreži G_f te mu dodjeljujemo kapacitet $c_f(v, u) = f(u, v)$. Ovaj brid označava da možemo poništiti do $f(u, v)$ jedinica toka kroz brid (u, v), to jest, puštanjem pozitivnog toka kroz suprotan brid (v, u) smanjujemo tok kroz originalni brid (u, v).*

Primijetimo da svaki brid u mreži G_f ima strogo pozitivan kapacitet te da svaki brid u originalnoj mreži može rezultirati generiranjem najviše dva nova brida u rezidualnoj mreži. Dakle, rezidualna mreža je iste asimptotske veličine kao i zadana mreža.

Na slici 2.2 prikazana je rezidualna mreža za mrežu prikazanu na slici 1.2.

Važno je uočiti da, ako možemo pustiti tok kroz rezidualnu mrežu, tada možemo pustiti tu dodatnu količinu toka kroz originalnu mrežu. Ovu tvrdnju formalno možemo zapisati u lemi koja slijedi, no definirajmo najprije sumu dva toka.



Slika 2.2: Rezidualna mreža

Definicija 2.1.3. *Za tok f i tok f', njihova suma $f + f'$ je zbroj tokova kroz sve bridove. Vrijednost $f + f'$ jednaka je zbroju $|f| + |f'|$.*

Ako vrijedi $f'' = f + f'$, tada vrijedi $f''(u, v) = f(u, v) + f'(u, v)$

Lema 2.1.4. *Neka je f tok u mreži G te neka je f' tok u mreži G_f. Tada je $(f + f')$ tok u G.*

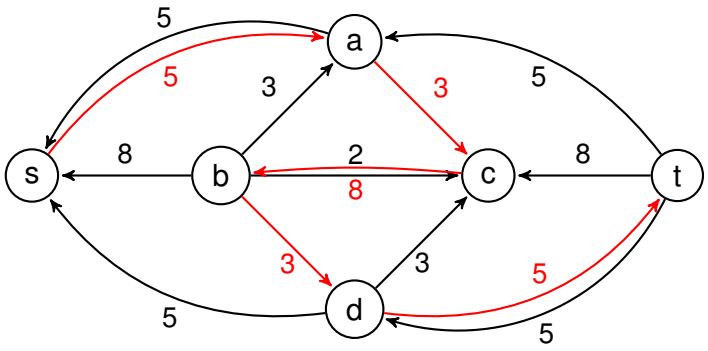
Lema implicira da sve što trebamo napraviti kako bismo povećali tok, je pronaći bilo koji tok u rezidualnoj mreži.

Sada definiramo povećavajuće puteve (eng. augmenting paths) na kojima povećavamo tok i rezidualni kapacitet.

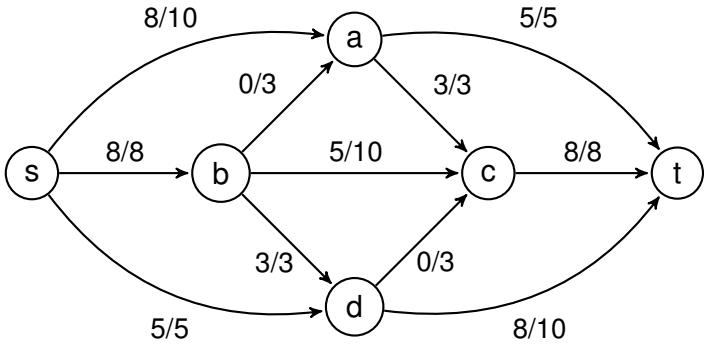
Definicija 2.1.5. *Neka je f tok kroz mrežu G, te neka je G_f pripadna rezidualna mreža. Povećavajući put je jednostavan put P od vrha s do vrha t u mreži G_f. Rezidualni kapacitet ili kapacitet uskog grla puta P je minimalan kapacitet bilo kojeg brida puta P. Označavamo ga s $c_f(P)$.*

Primjer 2.1.6. *Na slici 2.3 prikazan je jedan povećavajući put kapaciteta* 3*, za rezidualnu mrežu prikazanu na slici 2.2. Na slici 2.4 imamo prikazan graf nakon dodavanja toka svakom bridu puta. Obratimo pažnju na brid unatrag* (c, b)*, kod kojeg smo, zapravo, umanjili tok kroz brid* (b, c) *za* 3*.*

Puštanjem $c_f(P)$ jedinica toka kroz svaki brid puta, dobivamo valjani tok u G_f, a prema prethodnoj lemi, dodavanjem tog toka toku f, dobivamo i valjani tok za G strogo veće vrijednosti.



Slika 2.3: Povećavajući put



Slika 2.4: Tok nakon povećanja

2.2 Opis Algoritma

Razlika između algoritma opisanog u prethodnoj sekciji i pohlepnog algoritma je u tome što pohlepni algoritam samo povećava tok u bridovima. Obzirom na to da povećavajući put može povećati tok u suprotnom bridu, zapravo možemo smanjiti tok na nekom bridu originalne mreže.

Dobili smo ideju kako bi trebao izgledati algoritam za pronalazak maksimalnog toka kroz mrežu. Krećemo s tokom vrijednosti 0, te uzastopno pronalazimo povećavajući put. Ponavljamo ovaj postupak, sve dok postoji put kroz koji možemo pustiti još jedinica toka. Time dobivamo Ford-Fulkersonov metodu (za pravi algoritam, fali još metoda izbora povećavajućeg puta).

U nastavku donosimo pseudokod Ford-Fulkersonovog algoritma.

Listing 2.1: Ford-Fulkersonov algoritam)

```
ford-fulkerson-flow (G=(V, E, s, t)) {
        f=0
        while(true) {
                G' = rezidualna mreza od G za f
                if(G' nema s-t povecavajuci put)
                        break
                P = bilo koji povecavajuci put od G'
                c = najmanji kapacitet brida u putu P
                povecaj f tako da dodas c toku svakog brida u P
        }
        return f
}
```

Razmotrimo sada sljedeća pitanja:

1. Kako učinkovito možemo napraviti povećanje toka?
2. Koliko puta trebamo povećati tok dok ne dođemo do kraja algoritma?
3. Ako ne možemo napraviti više niti jedno povećanje, jesmo li pronašli maksimalan tok?

Prije nego odgovorimo na prvo pitanje, definirajmo red veličine neke funkcije (sljedeća definicija je preuzeta iz [8]).

Definicija 2.2.1. *Neka su $f, g : D \to \mathbb{R}$ dvije funkcije na odozgo neograničenom podskupu $D \subseteq \mathbb{R}$.*

1. *f nije većeg reda veličine od g, ili f ne raste brže od g, u oznaci $f(x) = O(g(x))$, ako $\exists C \in \mathbb{R}$ i $\exists x_0 \in D$, takvi da je $\forall x > x_0$, $|f(x)| < C|g(x)|$.*
2. *f nije manjeg reda veličine od g, ili f raste barem jednako brzo kao i g, u oznaci $f(x) = \Omega(g(x))$, ako $\exists \varepsilon > 0$ i $\exists$ niz $x_n \in D$, $n \in \mathbb{N}$, $x_n \to \infty$, takvi da je $\forall n \in \mathbb{N}$, $|f(x_n) > \varepsilon|g(x_n)|$.*

Sada, za zadane G i f, trebamo izračunati rezidualnu mrežu G_f. To lako možemo učiniti u $O(n + m)$ vremenu, pri čemu je $n = |V|$ i $m = |E|$. Pretpostavljamo da G_f sadrži samo bridove sa strogo pozitivnim kapacitetom. Zatim, trebamo odrediti postoji li povećavajući put od vrha s do vrha t u mreži G_f. To možemo učiniti pomoću pretraživanja u dubinu ili pretraživanja u širinu u rezidualnoj mreži, tako da krenemo od vrha s i stanemo čim dođemo do vrha t. Pretraživanje, također, možemo obaviti u vremenu $O(n + m)$. Neka je P put kojeg dobijemo kao rezultat. Konačno, tražimo brid s najmanjim kapacitetom kojeg sadrži put P te povećavamo tok f za tu veličinu, za svaki brid u putu P.

Prije nego odgovorimo na preostala dva pitanja, moramo definirati pojam reza.

2.3 Minimalan rez

Tok kroz mrežu ne možemo povećavati beskonačno, zbog toga što postoji podskup bridova kod kojih dolazi do zasićenja kapaciteta. Svaki put od vrha s do vrha t mora proći barem jednim od tih zasićenih bridova. Stoga, ukupna suma kapaciteta ovih bridova definira gornju granicu maksimalnog toka. Drugim riječima, ti bridovi čine usko grlo toka kroz mrežu. Takav skup bridova nalazi se na svakom putu od vrha s do vrha t. Možemo definirati particiju, koja odvajaja vrhove do kojih možemo doći iz vrha s, od vrhova do kojih ne možemo doći iz vrha s. Formalnije, rez definiramo u sljedećoj definiciji.

Definicija 2.3.1. *Za zadanu mrežu G, definiramo rez, ili s-t rez, kao particiju skupa vrhova na dva disjunktna podskupa $X \subseteq V$ i $Y = V \setminus X$, pri čemu je $s \in X$ i $t \in Y$. Tok kroz mrežu od X do Y definiramo kao razliku sume tokova od X do Y i sume tokova od Y do X.*

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y) - \sum_{y \in Y} \sum_{x \in X} f(y, x).$$

Primijetimo da je $f(X, Y) = -f(Y, X)$. Definirajmo još kapacitet reza.

Definicija 2.3.2. *Kapacitet reza (X, Y) je suma kapaciteta u bridovima koji vode od X do Y, odnosno,*

$$c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y).$$

Primijetimo da smo zanemarili kapacitet bridova koji vode od Y do X. Očito nije moguće pustiti više toka kroz rez, od samog kapaciteta reza.

Primjer 2.3.3. *Na slici 2.5 prikazan je tok vrijednosti 17 i rez $(X, Y) = (\{s, a\}, \{b, c, d, t\})$. pri čemu je $f(X, Y) = 17$.*

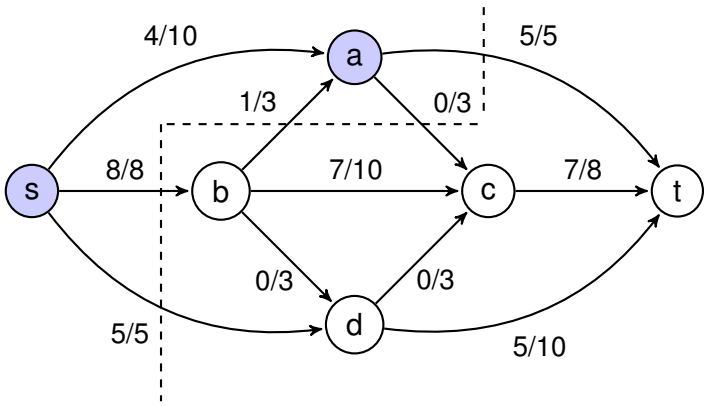
$$X = \{s, a\}, \quad Y = \{b, c, d, t\}, \quad f(X, Y) = 5 + 0 - 1 + 8 + 5 = 17.$$

2.4 Teorem maksimalnog toka

Kako bismo dokazali teorem na kojem se temelji optimalnost Ford-Fulkersonovog algoritma, potrebne su nam sljedeće dvije leme.

Lema 2.4.1. *Neka je (X, Y) s-t rez u mreži. Za svaki tok f, vrijednost toka f je jednaka toku kroz rez, odnosno, $f(X, Y) = |f|$.*

Lema 2.4.2. *Za svaki s-t rez (X, Y) i za svaki tok f, vrijedi $|f| \leq c(X, Y)$.*



Slika 2.5: Rez

Sada navodimo teorem poznat pod nazivom Teorem maksimalnog toka ili teorem najmanjeg reza. Teorem kaže da se u svakoj mreži minimalan kapacitet reza ponaša kao usko grlo koje ograničava maksimalnu količinu toka. Ford-Fulkersonov algoritam staje kada pronađe usko grlo te, na samom kraju, pronalazi oboje, minimalan rez i maksimalan tok.

Teorem 2.4.3. *Slijedeća tri uvjeta su ekvivalentna.*

1. *f je maksimalan tok u mreži G.*
2. *Rezidualna mreža G_f ne sadrži povećavajući put.*
3. *$|f| = c(X, Y)$, za neki rez (X, Y) u G.*

Dokaz. Dokazujemo, redom, sljedeće tri implikacije.

1. $1 \implies 2$: Dokazujemo kontradikcijom. Ako je f maksimalan tok te ako postoji povećavajući put u mreži G_f, kad bismo pustili tok kroz taj put, dobili bismo veći tok od f. Dakle, dobili smo kontradikciju.
2. $2 \implies 3$: Ako ne postoji povećavajući put, tada vrhovi s i t nisu spojeni u rezidualnoj mreži. Neka je X skup vrhova do kojih možemo doći iz vrha s u rezidualnoj mreži i neka skup Y sadrži sve ostale vrhove. Očito, (X, Y) čini rez. Obzirom da svaki brid, kojeg siječe rez, mora biti zasićen tokom, slijedi da je tok kroz rez jednak kapacitetu reza, dakle, $|f| = c(X, Y)$.
3. $3 \implies 1$: Ponovno dokazujemo kontradikcijom. Pretpostavimo da postoji tok f' čija vrijednost prelazi $|f|$. Iz toga bi slijedilo $|f'| > c(X, Y)$, što je u kontradikciji s lemom 2.4.2.

□

Ovim teoremom pokazali smo da Ford-Fulkersonov algoritam u konačnici daje maksimalan tok. No, staje li algoritam u konačno mnogo koraka i kolika je njegova vremenska složenost — to razmatramo u sljedećoj sekciji.

2.5 Složenost Ford-Fulkersonovog algoritma

Objasnimo najprije kako znamo da algoritam završava u konačno mnogo koraka. Prisjetimo se da smo pretpostavili da su svi kapaciteti bridova cijeli brojevi. Svako povećanje koje izvedemo, povećava tok za neki cijeli broj. Maksimalan broj povećanja je jednak sumi kapaciteta svih bridova koji su incidentni s vrhom s. Stoga, nakon konačnog broja povećanja, algoritam mora stati.

Ranije smo definirali $n = |V|$ i $m = |E|$. Obzirom da smo pretpostavili kako do svakog vrha možemo doći iz vrha s, slijedi da je $m \geq n - 1$. Dakle, $n = O(m)$. Sada vremensku složenost $O(n + m)$ možemo izraziti kao $O(m)$.

Kao što smo već naveli, rezidualnu mrežu možemo kreirati u vremenu $O(n+m) = O(m)$. Povećavajući put, također, možemo pronaći u vremenu $O(m)$. Iz toga slijedi da za svaki korak povećavanja trebamo $O(m)$ vremena. Postavlja se pitanje koliko mnogo povećavanja trebamo?

Nažalost, taj broj može biti jako velik. Ako s F^* označimo vrijednost maksimalnog toka kojeg trebamo dobiti, ukupan broj povećavajućih koraka može biti jednak F^*. Ako uzmemo da je svaki povećavajući korak vremenske složenosti $\Omega(m)$, ukupna vremenska složenost bi bila $\Omega(F^*m)$. Obzirom da F^* ne ovisi niti o m, niti o n, mogao bi biti proizvoljno velik, pa bi i vrijeme izračunavanja moglo biti proizvoljno veliko.

Možemo zaključiti da, ako ne odaberemo povećavajući put na pravi način, možemo uvelike povećati složenost algoritma.

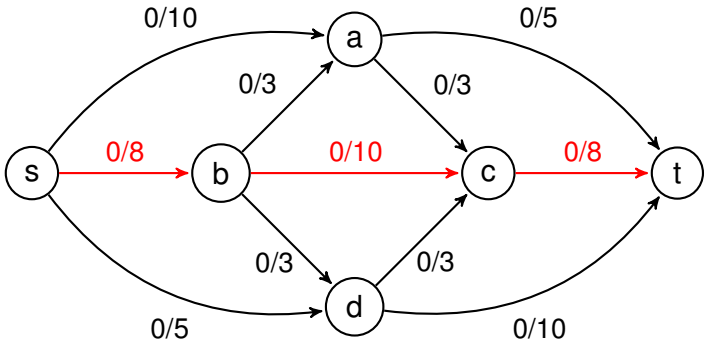
Postoje mnoge varijacije Ford-Fulkersonovog algoritma, koje rezultiraju mnogo boljom vremenskom složenošću. Neki od tih algoritama su Edmonds-Karpov algoritam, vremenske složenosti $O(nm^2)$, Dinitzev algoritam, složenosti $O(n^2m)$, King, Rao i Trajanov algoritam iz 1994. godine, vremenske složenosti $O(nm \log_{m/n \log n} n)$, te danas najbrži Orlinov algoritam iz 2012. godine, vremenske složenosti $O(nm)$.

U sljedećem poglavlju rada opisujemo Edmonds-Karpov algoritam.

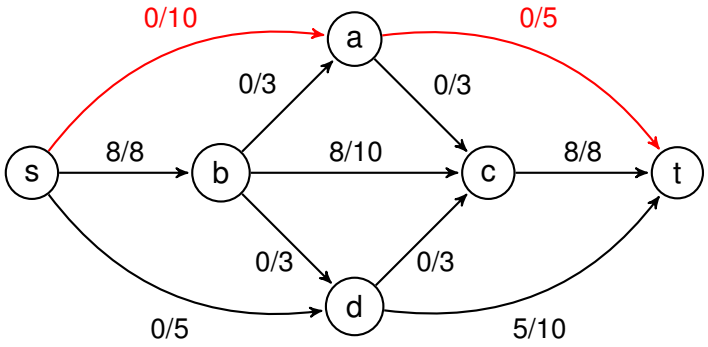
2.6 Primjer Ford-Fulkersonovog algoritma

Pokažimo na primjeru kako radi Ford-Fulkersonov algoritam za mrežu sa slike 1.1.

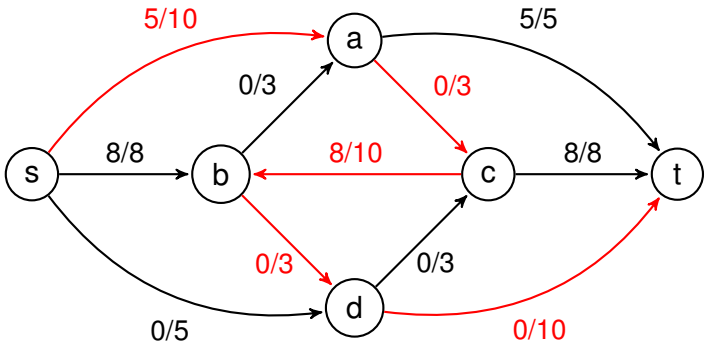
Primjer 2.6.1. *Najprije odaberemo jedan put u grafu, kako je prikazano na slici 2.6. Zatim, uvećavamo tok za najmanji kapacitet brida u putu. Sada ukupan tok kroz mrežu iznosi* $|f| = 8$. *Na slici 2.7 pronalazimo drugi put u grafu i povećavamo tok za* 5 *jedinica. Na slici 2.8 je prikazano stanje nakon uvećanja toka za* 5 *jedinica. Ukupan tok kroz mrežu sada iznosi* $|f| = 13$. *Označimo još jedan put kroz graf. Primijetimo da smo sada označili i brid* (c, b). *Kroz brid* (c, b) *šaljemo* 3 *jedinice toka, odnosno, umanjujemo tok kroz brid* (b, c) *za* 3 *jedinice toka. Sada ukupan tok kroz mrežu iznosi* $|f| = 16$. *Na slici 2.9 označavamo posljednji put kroz graf. Na slici 2.10 vidimo da više ne možemo poslati tok kroz niti jedan put. Maksimalan tok kroz mrežu koji smo pronašli iznosi* $|f| = 21$ *jedinica toka.*



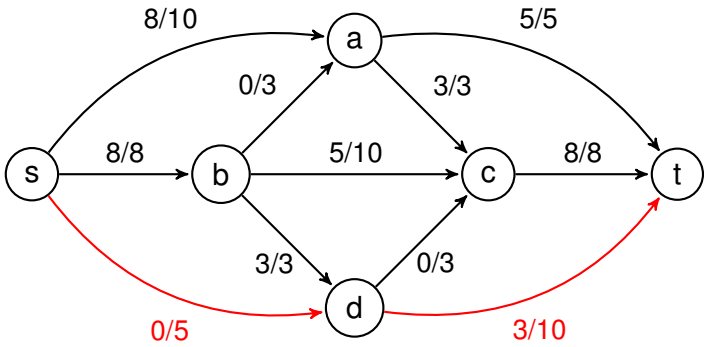
Slika 2.6: Odabir prvog puta



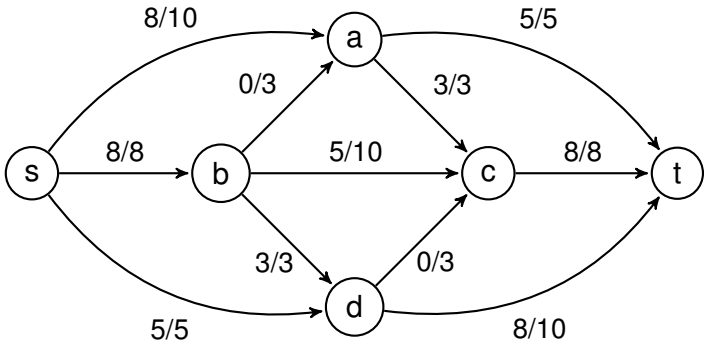
Slika 2.7: Odabir drugog puta



Slika 2.8: Odabir trećeg puta



Slika 2.9: Četvrti odabir puta



Slika 2.10: Kraj izračunavanja

Poglavlje 3

Edmonds-Karpov algoritam

3.1 Opis Algoritma

Jack R. Edmonds i Richard M. Karp prvi su razvili algoritam za problem toka kroz mrežu koji radi u polinomnom vremenu. Edmond-Karpov algoritam, objavljen 1972. godine, zapravo se temelji na Ford-Falkersonovom algoritmu, s time da prilikom pronalaska povećavajućeg puta, uzima onaj *s-t* put u rezidualnoj mreži koji sadrži najmanje bridova. Takav način rada može se postići korištenjem pretraživanja u širinu za računanje povećavajućeg puta. Pretraživanje u širinu efikasno pronalazi najkraći put na osnovu broja bridova.

Iz pseudokoda algoritma, također se vidi sličnost s Ford-Fulkersonovim alogritmom.

Listing 3.1: Edmonds-Karpov algoritam

```
Edmonds-Karp (G=(V, E, s, t)) {
        f=0
        while(true) {
                G' = rezidualna mreza od G za f
                if(G' nema s-t povecavajucih puteva)
                        break
                P = povecavajuci put za G' s najmanjim brojem bridova.
                c = najmanji kapacitet brida u putu P
                povecaj f tako da dodas c toku kroz svaki brid od P
        }
        return f
}
```

3.2 Složenost Edmonds-Karpovog algoritma

Općenito, pretraživanje u širinu grafa s n vrhova i m bridova zahtijeva $O(m+n)$ vremena. Kako smo već pretpostavili da vrijedi $m \geq n-1$, možemo pisati $O(m+n) = O(m)$. Dakle,

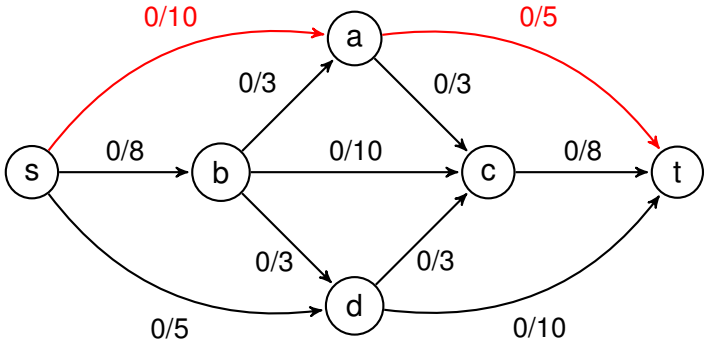
za pronalazak jednog puta P trebamo $O(m)$ koraka. Nakon toga, trebamo $O(n)$ koraka kako bismo povećali f i $O(n)$ koraka da konstruiramo G'. Ponovno koristimo činjenicu da vrijedi $n = O(m)$ te zaključujemo da svaki povećavajući put može biti izračunat u vremenu $O(m)$. Može se pokazati da, ako koristimo ovu metodu, ukupan broj povećavajućih koraka iznosi $O(nm)$. Iz navedenog slijedi da je ukupna vremenska složenost Edmonds-Karpovog algoritma jednaka $O(nm^2)$.

Više o složenosti Edmonds-Karpovog algoritma možete pronaći na [2].

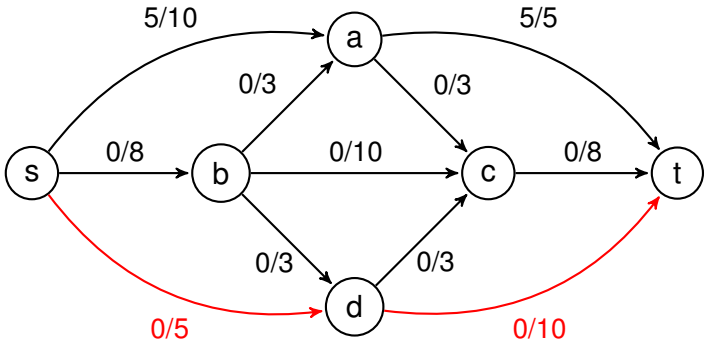
3.3 Primjer Edmonds-Karpovog algoritma

Pokažimo na primjeru kako radi Edmonds-Karpov algoritam za mrežu sa slike 1.1.

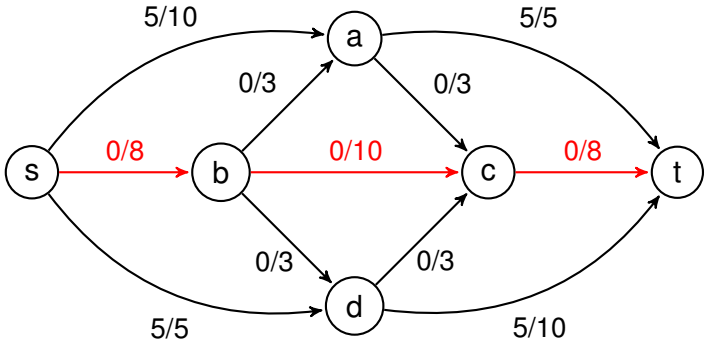
Primjer 3.3.1. *Najprije odaberemo jedan put s najmanjim brojem bridova u grafu, kako je prikazano na slici 3.1. Zatim, uvećavamo tok za najmanji kapacitet brida u tom putu. Sada ukupan tok kroz mrežu iznosi* $|f| = 5$. *Na slici 3.2 pronalazimo drugi put u grafu s najmanjim brojem bridova i povećavamo tok za* 5 *jedinica. Na slici 3.3 je prikazano stanje nakon uvećanja toka za* 5 *jedinica. Ukupan tok kroz mrežu sada iznosi* $|f| = 10$. *Označimo još jedan put kroz graf s najmanjim brojem bridova. Sada ukupan tok kroz mrežu iznosi* $|f| = 18$. *Na slici 3.4 označavamo jedini put kroz graf, kroz koji možemo pustiti tok. Primijetimo da smo i ovdje označili brid* (c, b), *kroz koji puštamo* 3 *jedinice toka, to jest umanjujemo tok kroz brid* (b, c) *za* 3 *jedinice toka. Na slici 3.5 vidimo da više ne možemo poslati tok kroz niti jedan put. Maksimalan tok kroz mrežu kojeg smo pronašli iznosi* $|f| = 21$ *jedinica toka.*



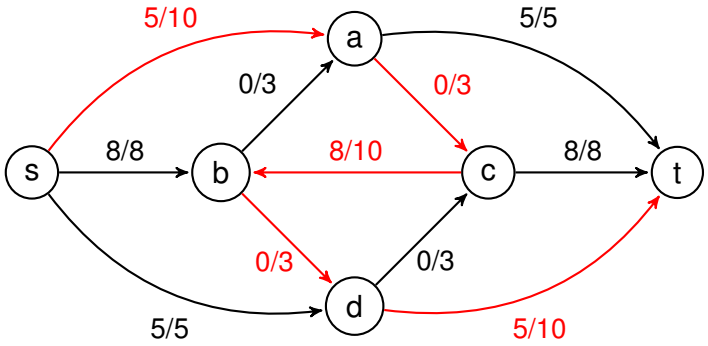
Slika 3.1: Odabir prvog puta



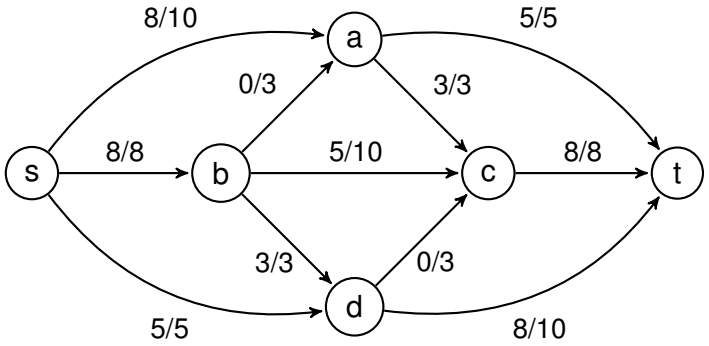
Slika 3.2: Odabir drugog puta



Slika 3.3: Odabir trećeg puta



Slika 3.4: Odabir četvrtog puta



Slika 3.5: Kraj izračunavanja

3.4 Edmonds-Karpov algoritam na računalu

Probajmo izračunati maksimalan tok u mreži iz prethodnog primjera pomoću Edmonds-Karpovog algoritma na računalu. Za implementaciju algoritma koristimo programski jezik Java. Program se bazira na pseudokodu navedenom u listingu 3.1. Prije nego pokrenemo program s našim primjerom, recimo samo da je za pronalaženje povećavajućeg puta u grafu korištena metoda pretraživanja u širinu, koja automatski pronalazi put s najmanjim brojem bridova. Donosimo pseudokod navedene metode, prilagođen za potrebe našeg programa.

Listing 3.2: Metoda pretraživanja u širinu

```
bfs (izvor, ponor, graf) {
        putPronaden = false;
        postavi sve vrhove na "neposjecene";
        q = new Queue();
        queue.add(izvor);
        roditelj[izvor] = -1;
        posjecen[izvor] = true;
        while (q != empty) do {
                x = q.dequeue();
                y = 1;
                while (y <= brVrhova) {
                        if( graf[x][y]>0 && !posjecen[y]) {
                                roditelj[y]=x;
                                queue.add(y);
                                posjecen[y]=true;
                        }
                        y++;

                }
        }
```

```
        if(posjecen[ponor])
                putPronaden = true;

        return putPronaden;

}
```

Za pokretanje programa trebamo još matricu koja prikazuje vrhove i kapacitete bridova, za naš primjer sa slike 1.1

$$M = \begin{bmatrix} 0 & 10 & 8 & 0 & 5 & 0 \\ 0 & 0 & 0 & 3 & 0 & 5 \\ 0 & 3 & 0 & 10 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 3 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Ako unesemo matricu M u naš program, dobivamo različit izbor puteva od onih u primjeru 3.3.1. Naime, računalo odabire sljedeće puteve:

$$s \to a \to t$$
$$s \to d \to t$$
$$s \to a \to c \to t$$
$$s \to b \to c \to t$$
$$s \to b \to d \to t$$

s vrijednostima toka 5, 5, 3, 5, 3.

Naravno, dobili smo isti krajnji rezultat. Maksimalan tok kroz mrežu iznosi $|f| = 21$.

Poglavlje 4

Proširene verzije problema toka kroz mrežu

Problem toka kroz mrežu od velike je važnosti zbog svoje primjene kod rješavanja velikog broja problema. Opisujemo dva korisna proširenja ovog problema. Navest ćemo kako se ti problemi mogu reducirati na problem toka kroz mrežu te kako možemo iskoristiti isti algoritam za rješavanje oba problema. Mnogi problemi izračunavanja, koji na prvi pogled nemaju veze s tokom tvari kroz mrežu, mogu se svesti na jednu od ove dvije proširene verzije.

Prilikom definiranja proširenih verzija problema toka kroz mrežu oslanjamo se na [1].

4.1 Prvo proširenje: Cirkulacije sa zahtjevima

Primijetimo da u našoj osnovnoj definiciji problema toka kroz mrežu imamo samo jedan izvor s i jedan ponor t. No, što ako imamo skup S izvora, koji generiraju tok, i skup T ponora, koji mogu apsorbirati isti taj tok?

Kod ovakvog problema, teško je odlučiti koji izvor ili ponor uzeti u obzir prilikom pronalaska maksimalnog toka. Stoga, umjesto povećavanja vrijednosti toka, promatramo problem kod kojeg izvori imaju fiksne vrijednosti opskrbe (eng. supply values), dok ponori imaju fiksne vrijednosti zahtjeva (eng. demand values). Cilj je poslati tok iz čvorova s opskrbnim vrijednostima prema onima sa zadanim zahtjevima. U ovom slučaju, ne želimo maksimizirati određenu vrijednost, već želimo zadovoljiti sve zahtjeve, koristeći dostupne opskrbne vrijednosti.

Pretpostavimo da imamo mrežu $G = (V, E)$ s kapacitetima na bridovima. Svaki vrh $v \in V$ u mreži G povezujemo sa zahtjevom d_v. Ako je $d_v > 0$, to znači da vrh v ima zahtjev

d_v za tokom. Preciznije, vrh je ponor i želi primiti d_v jedinica toka više nego što šalje van. Ako je $d_v < 0$, to znači da vrh v ima zalihu $-d_v$. Vrh je izvor i želi poslati $-d_v$ jedinica toka više nego što prima. Ako je $d_v = 0$, vrh v nije ni izvor, ni ponor. Pretpostavljamo da su svi kapaciteti i zahtjevi cijeli brojevi. Sada formalno definiramo cirkulaciju sa zahtjevima.

Definicija 4.1.1. *Kažemo da je f cirkulacija sa zahtjevima $\{d_v\}$, ako je f funkcija koja pridružuje nenegativan realan broj svakom bridu i zadovoljava sljedeća dva uvjeta.*

1. *Uvjeti kapaciteta: Za svaki brid $e \in E$, vrijedi $0 \leq f(e) \leq c_e$.*
2. *Uvjeti zahtjeva: Za svaki vrh $v \in V$, vrijedi $f^{in}(v) - f^{out}(v) = d_v$.*

Sada, umjesto problema maksimizacije, razmatramo problem izvedivosti (eng. feasibility problem). Želimo znati postoji li cirkulacija koja zadovoljava navedene uvjete. Postoji jednostavan uvjet koji mora vrijediti kako bi postojala izvediva cirkulacija — ukupan zbroj svih zaliha mora biti jednak ukupnom zbroju svih zahtjeva.

Teorem 4.1.2. *Ako postoji izvediva cirkulacija sa zahtjevima $\{d_v\}$, tada vrijedi $\sum_v d_v = 0$.*

Iz teorema 4.1.2 znamo da vrijedi

$$\sum_{v:d_v>0} d_v = \sum_{v:d_v<0} -d_v.$$

Označimo prethodnu vrijednost s D. Navedimo još neke teoreme bitne za problem cirkulacije sa zahtjevima i njegovu vezu s problemom maksimalnog toka.

Teorem 4.1.3. *U mreži G postoji izvediva cirkulacija sa zahtjevima $\{d_v\}$, ako i samo ako maksimalan s^*-t^* tok u mreži G' ima vrijednost D. Ako su svi kapaciteti i zahtjevi u G cijeli brojevi te postoji izvediva cirkulacija, tada postoji cjelobrojna izvediva cirkulacija.*

Teorem 4.1.4. *Graf G ima izvedivu cirkulaciju sa zahtjevima $\{d_v\}$, ako i samo ako za sve rezove (A, B) vrijedi*

$$\sum_{v \in B} d_v \leq c(A, B).$$

Dokazi ovih teorema mogu se pronaći u [1].

4.2 Drugo proširenje: Cirkulacije sa zahtjevima i donjim granicama

Ovaj problem je, zapravo, poopćenje prethodno navedenog problema. Često kod primjena ne želimo samo zadovoljiti zahtjeve na određene čvorove, već želimo ”prisiliti” tok da

koristi određene bridove. Željeni učinak možemo postići tako da, osim uobičajenih gornjih granica koje predstavlja kapacitet svakog brida, dodamo i donje granice na bridove.

Promotrimo mrežu $G = (V, E)$ s kapacitetom c_e i donjom granicom l_e, za svaki brid e. Pretpostavljamo da za svaki brid e vrijedi $0 \leq l_e \leq c_e$. Kao i kod prethodnog problema, svaki čvor v, također, ima zahtjev d_v koji može biti ili pozitivan ili negativan. Pretpostavljamo da su svi zahtjevi, kapaciteti i donje granice cijeli brojevi. Zadane veličine imaju isto značenje kao u prethodnom problemu, dok donja granica l_e znači da vrijednost toka u bridu e mora iznositi najmanje l_e. U ovom slučaju, cirkulacija kroz mrežu mora zadovoljiti sljedeće uvjete:

1. Uvjeti kapaciteta: Za svaki brid $e \in E$, vrijedi $l_e \leq f(e) \leq c_e$.
2. Uvjeti zahtjeva: Za svaki vrh $v \in V$, vrijedi $f^{in}(v) - f^{out}(v) = d_v$.

Ponovno nas zanima postoji li izvediva cirkulacija koja zadovoljava navedene uvjete. Želimo svesti ovaj problem na problem pronalaženja izvedive cirkulacija sa zahtjevima, ali bez donjih granica. Već smo vidjeli da se problem pronalaženja izvedive cirkulacije sa zahtjevima može svesti na klasični problem pronalaska maksimalnog toka.

Znamo da kroz svaki brid e trebamo poslati barem l_e jedinica toka. Stoga, definiramo početnu cirkulaciju f_0 kao $f_0(e) = l_e$. Početna cirkulacija f_0 zadovoljava uvjete kapaciteta i za donje i za gornje granice. No, moguće je da ne zadovoljava sve uvjete zahtjeva. Označimo veličinu

$$f_0^{in}(v) - f_0^{out}(v) = \sum_{e\ into\ v} l_e - \sum_{e\ out\ of\ v} l_e$$

s L_v. Ako je $L_v = d_v$, tada smo zadovoljili uvjet zahtjeva na v. Ako nije, tada moramo nadrediti (eng. superimpose) cirkulaciju f_1 prije cirkulacije f_0. Time bismo riješili "neravnotežu" u v. Dakle, trebamo $f_1^{in}(v) - f_1^{out}(v) = d_v - L_v$. Postavlja se pitanje koliko nam je preostalo kapaciteta s kojim bismo mogli postići traženu vrijednost. Obzirom da smo već pustili l_e jedinica toka kroz svaki brid e, preostaje nam još $c_e - l_e$ jedinica toka.

Konstruiramo graf G' s istim vrhovima i bridovima, kapacitetima i zahtjevima, ali bez donjih granica. Neka kapacitet brida e bude $c_e - l_e$ te neka zahtjev na čvor v bude $d_v - L_v$.

Tvrdimo da je naša konstrukcija ekvivalentna početno zadanom grafu, ali bez donjih granica. Stoga možemo iskoristiti isti algoritam i za rješavanje ovog problema.

Navedimo teorem na koji ćemo se pozivati prilikom rješavanja problema provođenja nadzora i određivanja rasporeda letenja.

Teorem 4.2.1. *Izvediva cirkulacija u grafu G postoji, ako i samo ako postoji izvediva cirkulacija u grafu G′. Ako su svi zahtjevi, kapaciteti i donje granice u grafu G cijeli brojevi te postoji izvediva cirkulacija, tada postoji izvediva cirkulacija koja je cjelobrojna.*

Dokaz teorema može se pronaći u [1].

Poglavlje 5

Primjene problema toka kroz mrežu

Važnost problema toka kroz mrežu zapravo se ne temelji na modeliranju toka kroz mrežu, već u činjenici da mnogi netrivijalni kombinatorni problemi mogu biti riješeni u polinomnom vremenu, ako ih svedemo na problem pronalaženja maksimalnog toka ili minimalnog reza u usmjerenom grafu. U ovom poglavlju donosimo dva primjera kada je moguće neki problem svesti na problem toka kroz mrežu. Kod rješavanja sličnih problema, ponekad je potrebno traženje maksimalnog toka, a ponekad minimalnog reza. Dakle, i maksimalan tok i minimalan rez su podjednako važni alati u rješavanju mnogih problema. Primjeri koje navodimo su preuzeti iz [1].

5.1 Provođenje nadzora

Za početak ćemo navesti jedan jednostavan primjer, kojeg ćemo nazvati provođenje nadzora. Provođenje nadzora je zadatak koji često zahtijevaju mnoge tvrtke koje žele saznati povratnu informaciju od svojih potrošača. Zanima ih u kojoj mjeri su njihovi potrošači zadovoljni uslugom koju tvrtka pruža. Ovaj problem ilustrira kako se konstrukcija, koja se inače koristi pri rješavanju problema bipartitnih grafova, može prirodno iskoristiti u bilo kojem slučaju, gdje želimo oprezno uskladiti odluke na skupu koji sadrži više opcija. U ovom slučaju, to bi bilo sastavljanje upitnika usklađivanjem važnih pitanja namijenjenih populaciji potrošača.

Opis problema

Najveći problem u području prikupljanja podataka (data mining) je upravo istraživanje o uzorcima preferencija potrošača. Uzmimo tvrtku koja prodaje k proizvoda te posjeduje bazu podataka sa zapisima o kupovini velikog broj potrošača. Podatke prikuplja pomoću kartica kluba vjernosti. Tvrtka želi provesti nadzor slanjem posebno sastavljenih upitnika

određenoj grupi od n potrošača. Cilj istraživanja je saznati koje proizvode potrošači preferiraju. Istraživanje se temelji na sljedećim smjernicama.

1. Svaki potrošač će dobiti pitanja o određenom skupu proizvoda.
2. Potrošač može biti ispitan samo o proizvodima koje je kupio.
3. Kako bi upitnici bili informativni, ali ne predugi, da ne odbiju potrošače od ispunjavanja, svakog potrošača i treba pitati o c_i do c'_i proizvoda.
4. Kako bismo prikupili dovoljno podataka o svakom proizvodu, moramo ispitati od p_j do p'_j različitih potrošača o proizvodu j.

Formalno, problem provođenja nadzora sastoji se od bipartitnog grafa G čiji vrhovi predstavljaju potrošače i proizvode. Između potrošača i i proizvoda j nalazi se brid ako je potrošač i kupio taj proizvod j. Za svakog potrošača $i = 1, \ldots, n$, imamo ograničenja $c_i \leq c'_i$ na broj proizvoda o kojima može biti pitan. Za svaki proizvod $j = 1, \ldots, k$, imamo ograničenja $p_j \leq p'_j$ na broj različitih kupaca koje treba ispitati o proizvodu j. Postavlja se pitanje može li se napisati upitnik za svakog potrošača, tako da zadovoljava sve navedene uvjete.

Konstrukcija algoritma

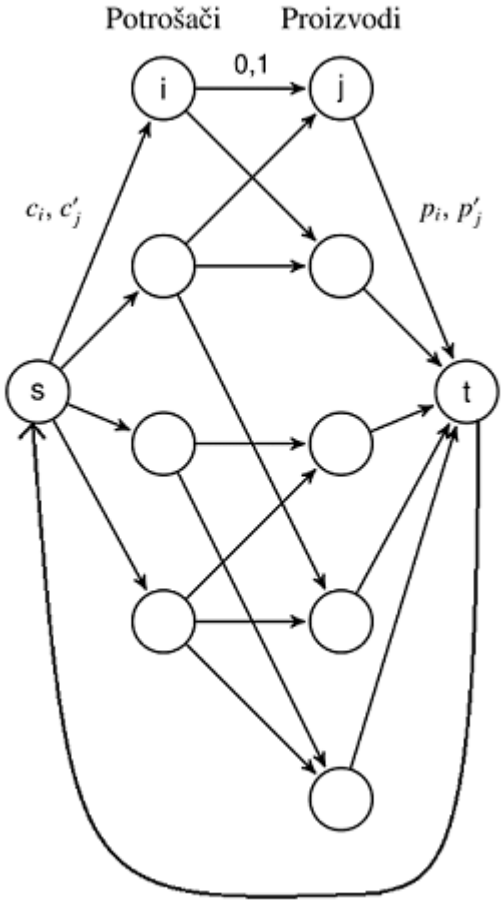
Problem rješavamo tako da ga svedemo na problem cirkulacije sa zahtjevima i donjim granicama u mreži G', kao na slici 5.1.

Kako bismo dobili graf G' iz grafa G, bridove u grafu G usmjeravamo od potrošača prema proizvodima. Dodajemo još vrhove s i t, s bridovima (s, i) za svakog potrošača $i = 1, \ldots, n$ i bridovima (j, t) za svaki proizvod $j = 1, \ldots, k$, te brid (t, s). Cirkulacija u ovoj mreži odgovara načinu na koji su postavljana pitanja. Tok kroz brid (s, i) je jednak broju proizvoda uključenih u upitnik namijenjen potrošaču i. Stoga, taj brid ima kapacitet c'_i i donju granicu c_i. Tok kroz brid (j, t) odgovara broju potrošača kojima je postavljeno pitanje o proizvodu j. Stoga, taj brid ima kapacitet p'_j i donju granicu p_j. Svaki brid (i, j), koji je usmjeren od potrošača ka proizvodu kojeg je taj potrošač kupio, ima kapacitet 1 i donju ogradu 0. Tok kroz brid (t, s) odgovara ukupnom broju postavljenih pitanja. Dakle, možemo mu dodijeliti kapacitet $\sum_i c'_i$ i donju granicu $\sum_i c_i$. Svi vrhovi imaju zahtjev 0.

Naš algoritam konstruira mrežu G' i provjerava postoji li izvediva cirkulacija.

Analiza algoritma

Teorem 5.1.1. *Graf G', konstruiran u gore opisanom algoritmu, ima izvedivu cirkulaciju, ako i samo ako postoji izvedivi (eng. feasable) način za konstrukciju nadzora.*



Slika 5.1: Provođenje nadzora

Dokaz. Prethodno opisana konstrukcija direktno sugerira način na koji bismo mogli provođenje nadzora pretvoriti u odgovarajući tok. Neka brid (i, j) prenosi jednu jedinicu toka ako je potrošaču i postavljeno pitanje o proizvodu j u nadzoru. Inače ne prenosi nikakav tok. Neka je tok kroz brid (s, i) broj pitanja postavljenih potrošaču i. Neka je tok kroz brid (j, t) broj potrošača kojima je postavljeno pitanje o proizvodu j, dok je tok kroz brid (t, s) ukupan broj postavljenih pitanja. Taj tok zadovoljava 0 zahtjeva, stoga vrijedi zahtjev očuvanja toka u svakom vrhu. Ako nadzor zadovoljava ova pravila, tada pripadni tok zadovoljava kapacitete i donje granice.

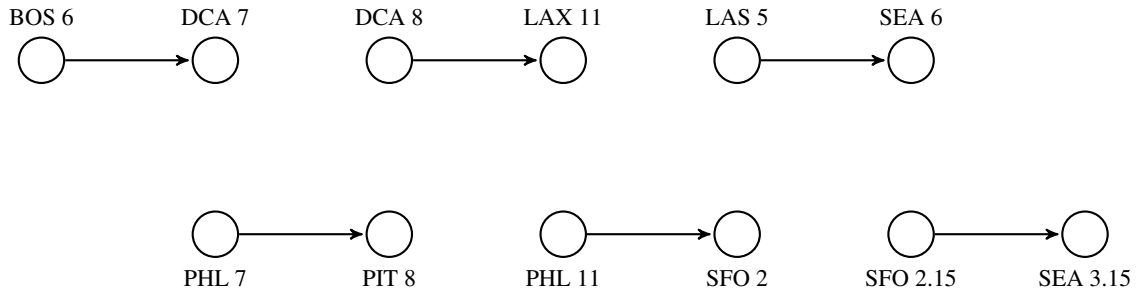
Obratno, ako je problem cirkulacije izvediv, tada po teoremu 4.2.1 postoji izvediva cjelobrojna cirkulacija, a takva cjelobrojna cirkulacija prirodno odgovara izvedivom provođenju nadzora (eng. feasible survey design). Potrošač i će biti ispitan (surveyed) o proizvodu j, ako i samo ako brid (i, j) prenosi jednu jedinicu toka. □

5.2 Raspored letenja

Problem izrade rasporeda letenja u stvarnom životu je jako kompliciran. Aviokompanije se susreću s mnogo zahtjeva. Broj ruta koje moraju uskladiti svaki dan mjeri se u tisućama. Također, moraju uskladiti posadu, održavanje zrakoplova te udovoljiti potrebama putnika. Osim toga, moraju voditi računa o nepredvidivim situacijama, poput vremenskih neprilika i kvarova. Mi svodimo ovaj problem na što jednostavniji primjer, koji bi nam mogao poslužiti kao temelj u rješavanju nekih drugih problema. Stoga ćemo zanemariti navedene komplikacije i rješavamo samo pojednostavljeni slučaj.

Opis problema

Pretpostavimo da je zadan skup od m letova. Jedan let j određen je s četiri parametra: aerodromom s kojeg polijeće, aerodromom na koji slijeće, vremenom polaska i vremenom dolaska. Na slici 5.2 je prikazan jednostavan primjer sa šest letova u jednom danu.



Slika 5.2: Raspored letenja (a)

Navedimo popis tih letova:

1. Boston (polazak 6 A.M.) – Washington DC (dolazak 7 A.M.)
2. Philadelphia (polazak 7 A.M.) – Pittsburgh (dolazak 8 A.M.)
3. Washington DC (polazak 8 A.M.) – Los Angeles (dolazak 11 A.M.)
4. Philadelphia (polazak 11 A.M.) – San Francisco (dolazak 2 P.M.)

5. San Francisco (polazak 2:15 P.M.) – Seattle (dolazak 3:15 P.M.)

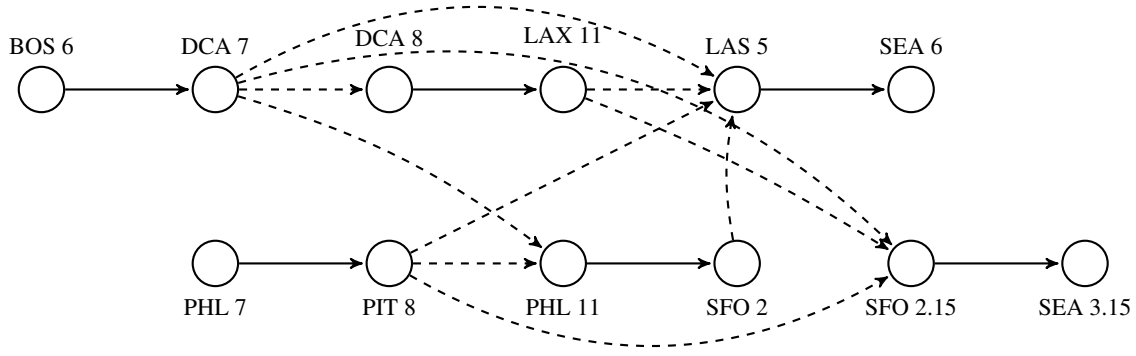
6. Las Vegas (polazak 5 P.M.) – Seattle (dolazak 6 P.M.)

Uočimo da je u letovima uključeno vrijeme polijetanja i slijetanja, kao i aerodromi polijetanja i slijetanja. Moguće je koristiti isti zrakoplov za let i i let j, ako osiguramo da vrijedi:

1. Mjesto dolaska leta i je upravo mjesto polaska leta j i ima dovoljno vremena za održavanje zrakoplova između ta dva leta.

2. Možemo dodati let između leta i i leta j, tako da zrakoplov dođe od mjesta dolaska leta i do mjesta polaska leta j u adekvatnom vremenu.

Tako bismo, na primjer, isti zrakoplov mogli koristiti za letove (1), (3) i (6). Taj zrakoplov bi najprije trebao pričekati u Washingtonu između letova (1) i (3), te ubaciti neki let između letova (3) i (6). Na slici 5.3 su prikazani takvi letovi.



Slika 5.3: Raspored letenja (b)

Formulirajmo problem. Situaciju ćemo modelirati kao jako općeniti slučaj. Samo ćemo reći da je let j dohvatljiv iz leta i, ako je moguće iskoristiti isti zrakoplov za let i i za let j. Zanemarit ćemo vrijeme potrebno za održavanje zrakoplova i profitabilnost pojedinog leta. Postavljanje problema uključuje i letove i specifikacije parova (i, j), pri čemu je kasniji let j dohvatljiv iz ranijeg leta i. Ti parovi mogu tvoriti proizvoljan usmjereni aciklički graf. Cilj je odrediti je li moguće održati svih m letova, koristeći najviše k zrakoplova. Znači, trebamo pronaći način da što efikasnije iskoristimo zrakoplove za što više letova.

Konstrukcija algoritma

Sada opisujemo algoritam za rješavanje problema rasporeda letenja, koji se temelji na problemu toka kroz mrežu. Neka jedinice toka predstavljaju zrakoplove. Svaki let ćemo prikazati pomoću jednog brida. Gornje i donje granice kapaciteta na tim bridovima postavljamo na 1. Time osiguravamo da na svakom bridu postoji točno jedna jedinica toka. Za svaki let postoji točno jedan zrakoplov. Neka brid (u_i, v_i) predstavlja let i i brid (u_j, v_j) predstavlja let j. Ako je let j dohvatljiv iz leta i, onda postoji brid iz vrha v_i u vrh u_j s kapacitetom 1. Na ovaj način, jedinica toka može proći kroz brid (u_i, v_i) i prijeći direktno u (u_j, v_j), kao što je prikazano na slici 5.3. Ako dodamo još vrhove izvora i ponora, dobivamo mrežu toka. Definirajmo formalno skup vrhova grafa G.

1. Za svaki let i, graf G ima dva vrha, u_i i v_i.
2. G ima izvor s i ponor t.

Definiramo skup bridova grafa G.

1. Za svaki let i, postoji brid (u_i, v_i) s donjom granicom 1 i kapacitetom 1. (Svaki let s popisa se mora održati.)
2. Za svaki let i i svaki let j, takve da je let j dohvatljiv iz leta i, postoji brid (v_i, u_i) s donjom granicom 0 i kapacitetom 1. (Možemo koristiti isti zrakoplov za letove i i j.)
3. Za svaki let i, postoji brid (s, u_i) s donjom granicom 0 i kapacitetom 1. (Svaki zrakoplov može započeti dan s letom i.)
4. Za svaki let j, postoji brid (v_j, t) s donjom granicom 0 i kapacitetom 1. (Svaki zrakoplov može završiti dan s letom j.)
5. Postoji brid (s, t) s donjom granicom 0 i kapacitetom k. (Ako imamo više zrakoplova koji nam nisu potrebni, ne moramo ih iskoristiti za niti jedan od letova.)

Vrh s zahtijeva $-k$ jedinica toka, a vrh t zahtijeva k jedinica toka. Svi ostali vrhovi zahtijevaju 0 jedinica toka.

Analiza algoritma

Dokazujemo da algoritam za određivanje rasporeda letenja radi korektno.

Teorem 5.2.1. *Postoji način da se održe svi letovi koristeći najviše k zrakoplova, ako i samo ako postoji izvediva cirkulacija u mreži G.*

Dokaz. Najprije pretpostavimo da postoji način da održimo sve letove, koristeći $k' \leq k$ zrakoplova. Skup letova izveden svakim pojedinim zrakoplovom definira put P u mreži G. Puštamo jednu jedinicu toka u svaki takav put P. Kako bismo zadovoljili potpune zahtjeve na vrhove s i t, puštamo $k - k'$ jedinica toka kroz brid (s, t). Dobivena cirkulacija zadovoljava sve zahtjeve kapaciteta i uvjeta donjih granica.

Obratno, promotrimo izvedivu cirkulaciju u mreži G. Iz teorema 4.2.1 znamo da postoji izvediva cirkulacija s cjelobrojnim vrijednostima toka. Pretpostavimo da je poslano k' jedinica toka kroz sve bridove, osim brida (s, t). Obzirom da svi ostali bridovi imaju vrijednost kapaciteta 1 i cirkulacija je cijeli broj, svaki takav brid koji prenosi tok sadrži točno jednu jedinicu toka.

Sada ćemo to pretvoriti u raspored, tako da pretvorimo tok u skup puteva. Promotrimo brid (s, u_i) koji prenosi jednu jedinicu toka. Prema zakonu očuvanja slijedi da brid (u_i, v_i) prenosi jednu jedinicu toka. Nastavimo li u ovom smjeru, možemo konstruirati put P od s do t, takav da svaki brid na tom putu prenosi jednu jedinicu toka. Taj postupak možemo primijeniti na svaki brid oblika (s, u_i), koji prenosi jednu jedinicu toka. Na ovaj način, dobili smo k' puteva od vrha s do vrha t, od kojih se svaki sastoji od bridova koji prenose po jednu jedinicu toka. Sada svakom putu P, kojeg smo kreirali na ovaj način, možemo pridružiti jedan zrakoplov, kako bismo održali sve letove koje taj put sadrži. □

Bibliografija

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest i C. Stein, *Introduction to Algorithms*, 2. izd., The MIT Press, Cambridge, Massachusetts, 2002.

[2] *CS 4820: Introduction to Analysis of Algorithms, Lecture notes on the Edmonds–Karp algorithm (Spring 2010)*, https://www.cs.cornell.edu/courses/cs4820/2012sp/handouts/edmondskarp.pdf, posjećeno 9. 9. 2018.

[3] *Ford–Fulkerson Algorithm for Maximum Flow Problem*, https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/, posjećeno 9. 9. 2018.

[4] *Maximum flow. Edmonds–Karp algorithm in* $O(\min(E^2 * V, E * FLOW))$, https://sites.google.com/site/indy256/algo/edmonds_karp, posjećeno 9. 9. 2018.

[5] D. M. Mount, *CMSC 451: Design and Analysis of Computer Algorithms (Lecture Notes, Spring 2015)*, https://www.cs.umd.edu/class/fall2015/cmsc451/Lects/cmsc451-fall15-lects.pdf, posjećeno 9. 9. 2018.

[6] J. B. Orlin, *Max flows in* $O(nm)$ *time, or better*, STOC '13 Proceedings of the forty-fifth annual ACM symposium on Theory of computing, Palo Alto, California, USA, June 1–4 2013, 2013, str. 765–774.

[7] *Java Program to Implement Ford-–Fulkerson Algorithm*, https://www.sanfoundry.com/java-program-implement-ford-fulkerson-algorithm/, posjećeno 9. 9. 2018.

[8] S. Singer, *Predavanja iz kolegija Oblikovanje i analiza algoritama*, http://degiorgi.math.hr/oaa/materijali/scans/pog_1.pdf, posjećeno 9. 9. 2018.

Sažetak

U ovom radu govorimo o problemu toka kroz mrežu. Najprije definiramo neke osnovne pojmove koji su nam potrebni za daljnje razmatranje problema. Potom opisujemo Ford-Fulkersonov algoritam, koji je jedan od najznačajnijih algoritama za rješavanje problema toka kroz mrežu. Analiziramo složenost algoritma te na primjeru pokazujemo kako se može izračunati maksimalan tok kroz mrežu pomoću ovog algoritma. Prije same analize, navodimo teorem maksimalnog toka, odnosno, minimalnog reza, koji govori o odnosu ta dva problema.

Nakon Ford-Fulkersonovog algoritma, opisujemo Edmonds-Karpov algoritam, koji je, zapravo, jedno poboljšanje Ford-Fulkersonovog algoritma.

Zatim uvodimo neka proširenja problema toka kroz mrežu, poput cirkulacija sa zahtjevima i cirkulacija sa zahtjevima i donjim granicama. Navedena proširenja poslužit će nam kod rješavanja problema provođenja nadzora i određivanja rasporeda letenja. Problem provođenja nadzora i problem određivanja rasporeda letenja samo su neke od primjena problema toka kroz mrežu.

Summary

In this paper we talk about the network flow problem. First of all, we define some of the basic terms needed for the future analysis of the problem. Then we describe the Ford-Fulkerson algorithm, which is one of the most important algorithms for solving the network flow problem. We analyse the complexity of the Ford-Fulkerson algorithm and also demonstrate by example how to compute the maximum flow by using this algorithm. Before the complexity analysis, we first talk about Max-Flow/Min-Cut Theorem, that talks about the relationship between maximum flow and minimum cut problems.

After describing Ford-Fulkerson's algorithm, we describe the Edmonds-Karp algorithm, which is, actually, one of improvements of the Ford-Fulkerson algorithm.

Finally, we introduce some extensions of the maximum flow problem, like circulations with demands and circulations with demands and lower bounds. Above mentioned extensions of the maximum flow will be used to solve the problem of survey design and airline scheduling. Survey design and airline scheduling are just some applications of the maximum flow problem.

Životopis

Rođena sam 12. veljače 1992. godine u Zadru. Osnovnu školu završavam u Ninu te upisujem gimnaziju Jurja Barakovića u Zadru. Nakon završetka srednje škole, 2010. godine upisujem Preddiplomski sveučilišni studij Matematika na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu. Preddiplomski studij završavam 2016. godine. Iste godine upisujem Diplomski studij Računarstvo i matematika na istom fakultetu, koji završavam ovim radom.