Problem maksimalnog protoka

Piplica, Jelena

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU

JELENA PIPLICA

PROBLEM MAKSIMALNOG PROTOKA

DIPLOMSKI RAD

Split, srpanj 2023.

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU

ODJEL ZA MATEMATIKU

PROBLEM MAKSIMALNOG PROTOKA

DIPLOMSKI RAD

Studentica:
Jelena Piplica

Mentor:
izv. prof. dr. sc. Jurica Perić

Split, srpanj 2023.

TEMELJNA DOKUMENTACIJSKA KARTICA

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU
ODJEL ZA MATEMATIKU

DIPLOMSKI RAD

PROBLEM MAKSIMALNOG PROTOKA

Jelena Piplica

Sažetak:

Dijkstra algoritam i Bellman-Ford algoritam su algoritmi koji se koriste za pronalaženje najkraćeg puta u težinskom grafu. Dijkstra algoritam djeluje na pozitivnim težinama bridova i koristi se za pronalaženje najkraćeg puta od jednog početnog vrha do svih ostalih vrhova u grafu. Bellman-Ford algoritam, s druge strane, može se koristiti čak i kada postoje negativne težine bridova, ali ima složenost vremena veću od Dijkstra algoritma. Ford-Fulkerson algoritam koristi se za pronalaženje maksimalnog mrežnog protoka, a temelji se na uvećavajućim putevima. Ovaj algoritam iterativno pronalazi uvećavajuće puteve i uvećava protok dok ne postigne maksimalni protok. Ovi algoritmi su važan alat za rješavanje različitih problema u području grafova i optimizacije.

Ključne riječi:

graf, digraf, luk, put, stablo, mreža, Dijkstra algoritam, Bellman-Ford algoritam, Ford-Fulkerson algoritam, težinska funkcija, minimalni rez, uvećavajući put, rezidualni graf, rezidualni kapacitet

Podatci o radu:

71 stranica, 29 slika, 0 tablica, 8 literaturnih navoda, jezik izvornika: hrvatski

Mentor: *izv. prof. dr. sc. Jurica Perić*

TEMELJNA DOKUMENTACIJSKA KARTICA

Članovi povjerenstva:

dr. sc. Ana Laštre, pred.

prof. dr. sc. Milica Klaričić Bakula

Povjerenstvo za diplomski rad je prihvatilo ovaj rad *14. srpnja 2023.*

TEMELJNA DOKUMENTACIJSKA KARTICA

FACULTY OF SCIENCE, UNIVERSITY OF SPLIT
DEPARTMENT OF MATHEMATICS

MASTER'S THESIS

MAXIMUM FLOW PROBLEM

Jelena Piplica

Abstract:

Dijkstra's algorithm and Bellman-Ford algorithm are algorithms used to find the shortest path in a weighted graph. Dijkstra's algorithm works with positive edge weights and is used to find the shortest path from a single source node to all other nodes in the graph. On the other hand, the Bellman-Ford algorithm can handle graphs with negative edge weights but has a higher time complexity compared to Dijkstra's algorithm. The Ford-Fulkerson algorithm is used to find the maximum flow in a network and is based on augmenting paths. This algorithm iteratively finds augmenting paths and increases the flow until it reaches the maximum flow. All algorithms are important tools for solving various graph and optimization problems.

Key words:

graph, digraph, arc, path, tree, network, Dijkstra's algorithm, Bellman-Ford algorithm, Ford-Fulkerson algorithm, weight function, minimum cut, augmenting path, residual graph, residual capacity

Specifications:

71 pages, 29 figures, 0 tables, 8 references, written in Croatian

Mentor: *izv. prof. dr. sc. Jurica Perić*

Committee:

dr. sc. Ana Laštre, pred.

TEMELJNA DOKUMENTACIJSKA KARTICA

prof. dr. sc. Milica Klaričić Bakula

This thesis was approved by a Thesis commettee on *July 14, 2023.*

Uvod

Problem pronalaska najkraćeg puta između početnog i krajnjeg vrha u grafu bitan je problem koji ima široku upotrebu u stvarnom svijetu. Modeliranjem stvari pomoću grafova primjena se proteže od navigacijskih sustava, prometnih simulacija pa sve do web pretraživanja, usmjeravanja podataka na internetu i upotrebe kod baza podataka. Iako je potreba za efikasnim algoritmima velika u primjeni se često upotrebljavaju heurističke metode koje na brz način daju približno dobra rješenja. Ako je potrebna preciznost problemu se pristupa egzaktnim algoritmima. Kao primjer u ovom radu su prikazani Dijkstra algoritam i Bellman-Ford algoritam. Najvažnijim dijelom rada smatramo problem mrežnog protoka koji predstavlja važnu skupinu optimizacijskih problema te je jedan od ključnih problema u operacijskim istraživanjima, računarstvu i kombinatorici. Posvetit ćemo se problemu pronalaska maksimalnog mrežnog protoka. Postoje brojni učinkoviti algoritmi za rješavanje ovog problema poput Ford-Fulkerson algoritma koji će isto tako biti predmet ovog rada.

Sadržaj

Poglavlje 1

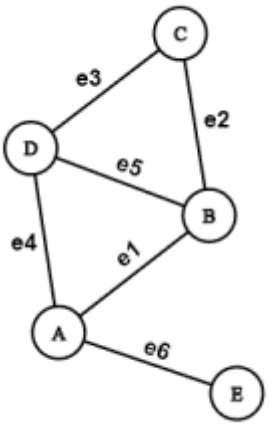
Algoritmi na grafovima

1.1 Osnovni pojmovi i definicije

Grafovi su jedna od najosnovnijih i najčešće korištenih matematičkih struktura. Koriste se za modeliranje brojnih problema i pojava u svakodnevnom životu. Prvim problemom u teoriji grafova smatra se problem sedam mostova Königsberga koji je 1736. godine postavio švicarski matematičar Leonhard Euler. Problem opisuje sedam mostova u Königsbergu koji povezuju kopno te se postavlja pitanje je li moguće pronaći put kojim ćemo proći kroz grad, a da svakim mostom prođemo točno jednom. Već iz ovog primjera možemo primijetiti da je problem traženja puta na karti kojom ćemo se kretati kako bi došli do željenog cilja povezan sa strukturom grafa. Općenito, traženje optimalnog puta od početne točke do željenog odredišta problem je s kojim se često susrećemo, od planiranja većih putovanja do svakodnevnice.

Definicija 1.1 *Graf je uređena trojka $G = (V, A, \phi)$, gdje je $V \neq \emptyset$ skup vrhova, a A skup bridova koji su disjunktni s V. ϕ predstavalja funkciju incidencije koja svakom bridu e pridružuje neuređeni par (ne nužno različitih) vrhova. $\phi(e) = \{u, v\}$, $u, v \in V$. Na taj način, svaki brid e spajat će dva ne*

nužno različita vrha $u, v \in V$ koje nazivamo krajevi od e. Kažemo da je brid e incidentan s vrhovima u, v, a te vrhove nazivamo susjednim vrhovima.



Slika 1.1: Graf $G = (V, A, \phi)$

Primjer 1.2 *Pogledajmo graf $G = (V, A, \phi)$ na Slici 1.1, gdje je $V = \{A, B, C, D, E\}$ s bridovima $A = \{e_1, e_2, e_3, e_4, e_5, e_6\}$. Funkcija incidencije tada bi bila $\phi(e_1) = \{A, B\}$, $\phi(e_2) = \{B, C\}$, $\phi(e_3) = \{C, D\}$, $\phi(e_4) = \{D, A\}$, $\phi(e_5) = \{D, B\}$, $\phi(e_6) = \{A, E\}$.*

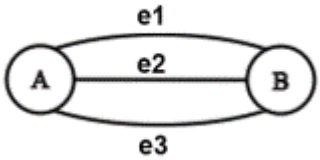
Napomena 1.3 *Graf G često označavamo i kao uređeni par $G = (V, A)$, pri čemu podrazumijevamo da su bridovi elementi svih dvočlanih podskupova skupa V, tj. $e = \{u, v\}$, $e \in A$, $u, v \in V$ ili skraćeno $e = uv$, $e \in A$, $u, v \in V$.*

Graf sadrži najmanje jedan vrh jer je skup vrhova V po definiciji neprazan skup, ali ne mora sadržavati bridove, tj. moguće je $A = \emptyset$. Graf koji ne sadrži bridove nazivamo prazan graf. Za graf koji sadrži samo jedan vrh kažemo da je trivijalan.

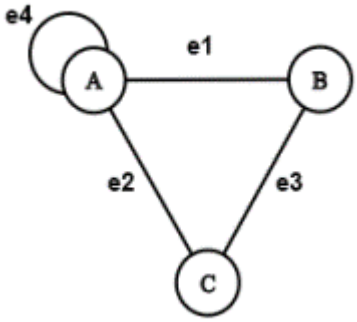
Funkcija incidencije ϕ ne mora biti injekcija pa je moguće da dva različita

vrha budu spojena s više bridova. Takvi bridovi se nazivaju višestruki bridovi.

Za $e \in A$ i $v \in V$ moguće je da je $\phi(e) = \{v, v\}$. Brid koji spaja vrh sa samim sobom nazivamo petlja.



Slika 1.2: Graf s višestrukim bridovima



Slika 1.3: Graf s petljom na vrhu A

Definicija 1.4 *Graf nazivamo jednostavnim ako nema petlji ni višestrukih bridova.*

Definicija 1.5 *Jednostavan graf kojemu je svaki par vrhova spojen bridom nazivamo potpun graf.*

Skupovi vrhova i bridova ne moraju biti konačni, u tom slučaju radi se o beskonačnom grafu. U ovom radu bavit ćemo se konačnim grafovima, odnosno

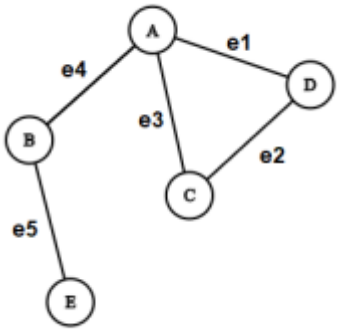
grafovima u kojima su skupovi vrhova i bridova konačni skupovi. Za takve grafove definirana su dva parametra: red i veličina grafa.

Definicija 1.6 *Red grafa $G = (V, A)$, u oznaci $|G|$ ili n, predstavlja broj vrhova u grafu G. Veličina od $G = (V, A)$, u oznaci $||G||$ ili m, predstavlja broj bridova u grafu G.*

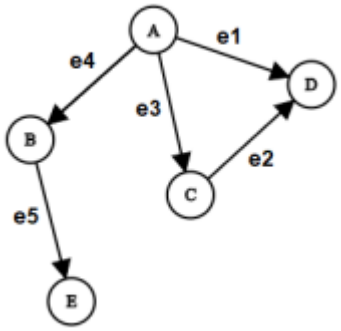
Ako je red grafa jednak n onda će veličina grafa biti izmedu 0 i $\binom{n}{2}$. Graf ima veličinu 0 u slučaju kada ne sadrži nijedan vrh, a $\binom{n}{2}$ u slučaju kada je graf potpuni neusmjereni graf s n vrhova.

Definicija 1.7 *Usmjeren graf ili digraf je uređena trojka $G = (V, A, \phi)$, gdje je $V \neq \emptyset$ skup vrhova, A skup bridova, a $\phi(G) : A \to V \times V$ funkcija incidencije koja svakom bridu $e \in A$ pridružuje uređeni par ne nužno različitih vrhova (u, v), $u, v \in V$. Vrh u tada zovemo početni vrh, a vrh v krajnji vrh, dok brid koji ima usmjerenje nazivamo luk.*

Primjer 1.8 *Promatramo dva grafa, $G = (V_1, A_1)$ i $D = (V_2, A_2)$ na Slikama 1.4 i 1.5. Oba grafa za skup vrhova imaju skup $\{A, B, C, D, E\}$. Razlikuju se po skupu bridova, tj. njihovoj (ne)uređenosti. Brid $e_1 \in G$ bit će neuređeni par vrhova $e_1 = \{A, B\}$, $A, B \in G$, dok će u grafu D brid koji spaja iste vrhove biti uređeni par $e_1 = (A, B)$, $A, B \in D$. Tako redom možemo proći sve bridove, npr. $e_2 = \{D, C\}$, $e_2 \in G$, $e_2 = (C, D)$, $e_2 \in D$, $e_3 = \{A, C\}$, $e_3 \in G$, $e_3 = (A, C)$, $e_3 \in D$... U slučaju poput ovoga, kada se skupovi vrhova poklapaju, a bridova razlikuju samo po uređenosti, graf D zovemo orijentacija grafa G, oznaka $D = \vec{G}$. Općenito, svaki neusmjereni graf možemo pretvoriti u usmjereni tako da odredimo početni i krajnji vrh svakog brida tj. da damo bridovima usmjerenje.*



Slika 1.4: Neusmjereni graf G



Slika 1.5: Usmjereni graf D

Šetnje u grafu nam omogućuju da istražujemo ponašanje i širenje informacija, pronalazimo povezanosti i rješavamo optimizacijske probleme.

Definicija 1.9 *Šetnja u grafu G je netrivijalan konačan niz* $W = v_0e_0v_1e_1...e_{k-1}v_k$ *vrhova i bridova u G takvi da* $e_i = \{v_i, v_{i+1}\}$ *za sve* $i < k$.

W tada zovemo šetnja od v_0 do v_k, često u oznaci (v_0, v_k). Vrhove v_0 i v_k nazivamo početak i kraj šetnje, a sve ostale vrhove u šetnji $v_1, ..., v_{k-1}$ nazivamo unutarnji vrhovi šetnje. Ako je $v_0 = v_k$, kažemo da je šetnja

zatvorena.
Broj k zovemo duljina šetnje W.

Definicija 1.10 *Neka je $W = v_0e_0v_1e_1...e_{k-1}v_k$ šetnja u grafu G. Ako su u šetnji W svi bridovi međusobno različiti, tada W zovemo staza. Ako su svi vrhovi u šetnji medusobno različiti onda šetnju W zovemo put.*

Definicija 1.11 *Put koji započinje i završava u istom vrhu zove se ciklus. Jednostavan ciklus je ciklus u kojem se samo početni i završni vrh ponavljaju.*

Definicija 1.12 *Za graf kažemo da je povezan ako postoji put između svaka dva vrha.*

Definicija 1.13 *Neka je $G = (V, A)$ graf. Za graf G kažemo da je stablo ako ne sadrži cikluse i izolirane vrhove te ako postoji put između svaka dva vrha iz V.*

Ako postoji put između svaka dva vrha u grafu, tada ne može biti izoliranih vrhova. Šuma je graf bez ciklusa, odnosno graf koji ne sadrži zatvorene puteve. Šuma se sastoji od skupa stabala, gdje je svako stablo neovisno i ne povezuje se s ostalim stablima u grafu.
Davanje težina bridovima u grafu je zanimljivo jer omogućava modeliranje situacija u kojima bridovi imaju različite karakteristike, kao što su udaljenosti, troškovi, vremenski zahtjevi ili druge vrijednosti koje se mogu mjeriti.

Definicija 1.14 *Neka je $G = (V, A)$ graf i $c : A \to \mathbb{R}$ funkcija koja svakom bridu iz grafa G pridružuje realan broj. Takvu funkciju zovemo težinska funkcija, a graf za koji je definirana takva funkcija težinski graf. Realan broj $c(e)$ pridružen bridu $e \in A$ zovemo težina brida e.*

Definicija 1.15 *Neka je $G = (V, A)$ težinski graf s težinskom funkcijom $c : A \to \mathbb{R}$ i neka je $p = v_0e_0v_1e_1...e_{k-1}v_k$ put u grafu G. Težina puta p definira se kao suma svih težina bridova koji su sadržani u putu:*

$$c(p) = \sum_{i=1}^{k-1} c(e_{i-1}, e_i).$$

Pronalaženje najkraćeg puta problem je koji je postavljen pri promatranju težinskih grafova. Postavlja se pitanje kako ćemo doći od jedne točke do druge na način da prijeđemo minimalnu udaljenost. Intuitivno, ako uzmemo u obzir sve moguće puteve između dvije odabrane točke, odabrat ćemo onaj put u kojem je suma svih udaljenosti između točaka na tom putu najmanja jer ćemo na taj način prijeći najmanju ukupnu udaljenost. Ovakav pristup može biti problematičan ako radimo s većim grafovima poput cestovne mreže nekog grada zbog tog što će broj mogućih puteva između dvije točke biti prevelik. Iz tog razloga, algoritamskim pristupom ovom problemu, razvile su se metode pronalaska najkraćeg puta koje ćemo promatrati u nastavku.

Definicija 1.16 *Težina najkraćeg puta $d(u, v)$ između vrhova $u, v \in V$ definira se s:*

$$d(u,v) = \begin{cases} \min\{c(p) \ : \ p \text{ put od } u \text{ do } v\}, & \text{ako postoji put od } u \text{ do } v \\ \infty, & \text{inače} \end{cases}$$

Može se dogoditi da težina najkraćeg puta nije dobro definirana ukoliko postoji ciklus čija je ukupna težina negativna. Sada ćemo razmotriti dva slavna algoritma za pronalazak najkraćeg puta u grafu kada su težine svih lukova nenegativne (Dijkstra algoritam), a zatim Bellman-Ford algoritam kada težine lukova mogu biti i negativne.

1.2 Dijkstra algoritam

Dijkstra algoritam, nazvan je po nizozemskom znanstveniku Edsgeru Dijkstri, razvijen je 1956. godine kao efikasan algoritam za pronalaženje najkraćeg puta. Dijkstra algoritam postao je ključni koncept u teoriji grafova i bio je temelj za razvoj kasnijih algoritama za pretraživanje najkraćeg puta. Kao što je prethodno napisano, ako je $c(i,j)$ nenegativan za sve $(i,j) \in A$ možemo koristiti Dijkstra algoritam. $d(i)$ je trenutna pretpostavka algoritma o težini najkraćeg puta od s do i koja je uvijek gornja granica udaljenosti najkraćeg puta. Vrhove označimo kad smo sigurni da je njihova udaljenost najmanja. U početku su svi vrhovi neoznačeni. Budući da algoritam održava udaljenost koja je gornja granica najmanje udaljenosti, najlakše je početi s $d(i) = \infty, \forall i \in V$. Međutim, udaljenost od s do s možemo postaviti na nulu. Budući da su sve težine luka nenegativne, ne može postojati put od s do s s težinom manjom od nule, stoga možemo postaviti $d(s) = 0$, i označiti vrh s. Sada možemo ažurirati udaljenosti za sve vrhove i u luku $(s,i) \in A$. Budući da znamo da je duljina najkraćeg puta od s do s nula, znamo da postoji put duljine najviše $d(s) + c(s,i) = c(s,i)$ od s do i (naime, put koji se sastoji od jednog luka (s,i)). Dakle, $d(s) + c(s,i)$ je gornja granica duljine najkraćeg puta od s do i pa onda možemo staviti $d(i) = min\{d(i), d(s) + c(s,i)\}$ za i tako da je $(s,i) \in A$. I dalje vrijedi da je $d(i)$ gornja granica na najkraćem $s - i$ putu. Ključ za Dijkstrin algoritam je da od svih neoznačenih vrhova možemo ispravno označiti onaj s minimalnom udaljenošću. Ako ima više vrhova s istom minimalnom udaljenošću onda možemo proizvoljno odabrati jedan. Dokažimo da je to točno. Pretpostavimo da vrh i ima minimalnu udaljenost $d(i)$ i označimo vrh i. Zatim kao i gore za sve lukove (i,j) znamo da postoji put duljine najviše $d(i) + c(i,j)$ do vrha j (koji se sastoji od najkraćeg $s - i$ puta nakon čega slijedi luk (i,j)). Zbog toga vrijedi

$d(j) = min\{d(j), d(i) + c(i,j)\}, \forall j$ tako da je $(i,j) \in A$. Zatim označimo neoznačeni vrh minimalne udaljenosti i ponovimo postupak. Primijetimo da se svaka udaljenost može samo smanjivati tijekom algoritma. Osim toga, iz prethodnog znamo da se put do vrha j sastoji od $s-i$ puta nakon čega slijedi luk (i,j). Stoga možemo pratiti trenutni put do j pokazivačem $p(j)$ na vrh i koji prethodi putu, a $p(j)$ ćemo nazvati roditeljem od j. Kada ažuriramo $d(j)$, ako postavimo $d(j) = d(i) + c(i,j)$, također postavljamo $p(j) = i$, tako da znamo da je trenutni put od s do j luk (i,j) dodan trenutnoj putanji od s do i. Da pronađemo put od s do j, počinjemo od j i pratimo roditeljske pokazivače od j natrag do s. Radi jednostavnosti postavili smo roditelja od s na nulu.

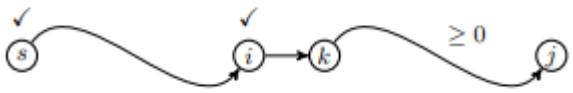
Dijkstra algoritam sažimamo u Algoritmu 1 i dokazat ćemo njegovu ispravnost.

Algorithm 1 Dijkstra algoritam za pronalazak najkraćeg puta

$d(i) \leftarrow \infty, \forall i \in V$
$p(i) \leftarrow null, \forall i \in V$
Unmark all $i \in V$
$d(s) \leftarrow 0$
while not all vertices are marked **do**
    Find unmarked $i \in V$ that minimizes $d(i)$ and mark i
    for j such that $(i,j) \in A$ **do**:
        if $d(j) > d(i) + c(i,j)$ **then**:
            $d(j) \leftarrow d(i) + c(i,j)$
            $p(j) \leftarrow i$
        end if
    end for
end while

Teorem 1.17 *Ako su sve težine lukova nenegativne, tada Dijkstra algoritam (Algoritam 1) određuje najkraću udaljenost od izvora s do svakog vrha $i \in V$.*

Dokaz. Dokaz ćemo provesti indukcijom. Kada algoritam označi vrh j, vrijednost $d(j)$ mora biti duljina najkraćeg puta od s do j. Prethodno smo dokazali da je $d(s) = 0$ i jasno je da je s prvi označeni vrh. Sada pretpostavimo da će neka iteracija algoritma označiti vrh $j, j \neq s$, algoritam je ispravno izračunao $d(i)$ za sve vrhove i prethodno označene algoritmom. Podsjećamo da je $d(j)$ gornja granica najmanje duljine $s-j$ puta tako da je $d(j)$ netočan samo ako postoji najkraći $s-j$ put P koji ima duljina strogo manju od $d(j)$. Pretpostavimo suprotno, da takav put P postoji. Slijedimo put P od s do j dok ne dođemo do zadnjeg vrha $i \neq j$ na stazi koja je označena, bit će neki takav vrh jer je s već označen, a j nije označen. Neka (i, k) bude luk izvan i na putu P, k nije označen, primijetimo da je moguće $k = j$. Po našoj pretpostavci indukcije budući da je i označen $d(i)$ je duljina najkraćeg puta od s do i. Nakon što smo označili i morao je biti slučaj da je $d(k) \leq d(i) + c(i, k)$: ili je to već bilo točno ili postavljamo $d(k) = d(i) + c(i, k)$ nakon označavanja i. Duljina ostatka puta P od k do j mora biti nenegativna jer su sve težine lukova nenegativne. $d(k)$ je donja granica duljine puta P koja je strogo manja od $d(j)$ po pretpostavci. Međutim, sada smo došli do kontradikcije jer je k neoznačen i ima udaljenost strogo manju od $d(j)$: ako je $k = j$ tada imamo $d(j) < d(j)$ ili ako je $k \neq j$ drugi neoznačeni vrh ima minimalnu udaljenost umjesto j. ■



Slika 1.6: Ilustracija dokaza Teorema 1.17. Ovo je najkraći $s - j$ put P duljine strogo manje od $d(j)$. Vrh i je zadnji označeni vrh na putu P, k je sljedeći vrh na putu i on mora biti neoznačen. Dokaz tvrdi da je tada $d(k)$ donja granica duljine puta pa je neki vrh osim j trebao biti sljedeći koji će biti odabran i označen.

Brzina izvršavanja programa ovisi o složenosti algoritma koji se izvršava. Ako je složenost mala, program će se brzo izvršiti čak i za velik broj elemenata. Ako je složenost velika program će se izvoditi polako ili čak neće raditi za velik broj elemenata.
Vremenska složenost Dijkstra algoritma u najgorem slučaju iznosi $\mathcal{O}(n^2)$. Koristeći podatkovnu strukturu *heap* smanjujemo složenost i ona će iznositi $\mathcal{O}(m \log n)$. *Heap* sadrži skup stavki, a svaka stavka ima pridruženu vrijednost koja je njen ključ. *Heap* kao struktura podataka podržava sljedeće operacije: *new heap*() koja vraća prazan *heap*, $h.insert(i, k)$ koja umeće stavku i u *heap* h s vrijednošću ključa k, $h.decrease - key(i, k')$ koja smanjuje ključ i na k'(pretpostavlja se da je k' manji od trenutnog ključa i), $h.extract - min()$ koja vraća stavku i minimalne vrijednosti ključa u *heapu* h i uklanja i iz *heapa*, *h.empty*? koji vraća *true* ako *heap* h nema stavki u sebi, a vraća $false$ u suprotnom. Stavke u *heapu* su vrhovi, a njihovi ključevi su udaljenosti. Primijetimo da označavanje vrhova zamjenjujemo nečlanstvom u *heapu*; ako vrh je u *heapu*, tada je neoznačen.
Heap je lako implementirati korištenjem polja. Najjednostavnija implementacija strukture podataka *heap* je složenosti $\mathcal{O}(1)$ za naredbu *new heap*,

za naredbu *insert* složenost je $\mathcal{O}(\log n)$ (s obzirom da umećemo najviše n stavki), za $decrease - key$ složenost je $\mathcal{O}(\log n)$, za naredbu $extract - min$ složenost je $\mathcal{O}(\log n)$, za naredbu *empty*? složenost je $\mathcal{O}(1)$. Ove složenosti za navedenu strukturu podataka daju ukupnu složenost $\mathcal{O}(m \log n)$ za Dijkstra algoritam budući da izvodimo n $extract - min$, n $inserts$ i najviše m $decrease - keys$ naredbi.

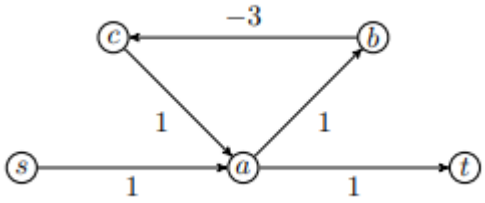
Algorithm 2 Dijkstra algoritam za pronalazak najkraćeg puta koristeći podatkovnu strukturu *heap*

```
h ← new heap();
d(i) ← ∞, ∀i ∈ V
p(i) ← null, ∀i ∈ V
d(s) ← 0
for i ∈ V do:
    h.insert(i,d(i))
end for
while not h.empty? do
    i ← h.extract − min()
    for j such that (i, j) ∈ A do
        if d(j) > d(i) + c(i, j) then
            d(j) ← d(i) + c(i, j)
            p(j) ← i
            h.decrease-key(j,d(j))
        end if
    end for
end while
```

1.3 Bellman-Ford algoritam

Sada se okrećemo slučaju u kojem težina luka može biti negativna. Teško se sjetiti slučajeva koji uključuju fizičko putovanje kroz mrežu gdje postoje lukovi negativne duljine. Međutim, pri modeliranju problema omogućit ćemo i negativne težine. Ovu ćemo situaciju susresti u Bellman-Ford algoritmu. Algoritam Bellman-Ford, kao i Dijkstra algoritam, traži najkraći put za jedan početni vrh u težinski usmjerenom grafu.

Jednom kada dopustimo lukove negativne težine moramo razmotrimo mogućnost da možda ne postoji $s - i$ put najkraće ukupne duljine. Za bilo koji broj B, možda postoji put duljine manje od B.



Slika 1.7: Negativna težina ciklusa na putu od s do t

Pogledajmo $s - t$ puteve sa Slike 1.7, put $(s, a), (a, t)$ ima težinu 2, put (s, a), (a, b), (b, c), (c, a), (a, t) ima težinu 1, put (s, a), (a, b), (b, c), (c, a), (a, b), (b, c), (c, a), (a, t) ima težinu 0, i tako dalje. Svaki put kad prijeđemo ciklus $a - b - c$ težina se smanjuje za 1. Kako bismo spriječili ovu mogućnost, zahtijevamo da i naš algoritam za problem najkraćeg puta pronađe najkraći put ili se završi jer postoji negativna težina ciklusa dostupna iz s.

Ciklus ima negativnu težinu ako je zbroj težina lukova u ciklusu negativan, dok je vrh i dohvatljiv iz s ako postoji put iz s do i, a ciklus je dohvatljiv iz s ako je bilo koji vrh u ciklusu dohvatljiv iz s.

Postoje jednostavni najkraći putevi od s do svakog $i \in V$ dostupni iz s ako i

samo ako ne postoje ciklusi negativne težine dohvatljivi iz s (sjetimo se da u jednostavnom putu nema vrha koji se ponavlja). Kako bismo donekle pojednostavili našu raspravu počinjemo s pretpostavkom da u ulaznom grafu nema ciklusa negativnih težina, a zatim pokazujemo kako to učiniti ako ih ima i kako ih otkriti. Kao i u Dijkstra algoritmu, ovaj algoritam održavat će skup udaljenosti $d(i), \forall i \in V$, gdje je inicijalno $d(s) = 0$ i $d(i) = \infty, \forall i \in V$, $i \neq s$. Algoritam će imati svojstvo da kad god je $d(i)$ konačan to je uvijek duljina nekog puta od s do i. Kad god je $d(j) > d(i) + c(i, j)$ možemo postaviti $d(j) = d(i) + c(i, j)$. Po našoj invarijanti postoji neki put do i duljine $d(i)$ i tako možemo pronaći kraći put do j koji prvi posjećuje i, a zatim koristi luk (i, j) da dođemo do j. Kao što je bio slučaj s Dijkstra algoritmom, mi također održavamo skup roditeljskih pokazivača $p(j)$ koji pokazuju na prethodni vrh trenutnog puta do j. Dakle, kada postavimo $d(j) = d(i) + c(i, j)$, znamo da je put do j došao iz vrha i i postavili smo $p(j) = i$. Promatrajući udaljenosti vidimo da se smanjuju tijekom algoritma. Glavna je ideja analize pokazati da nakon provjere svih lukova k puta imamo točno pronađene sve najkraće puteve koji koriste najviše k lukova. Dakle, pod pretpostavkom da ne postoji ciklus negativne težine, algoritam završava nakon $n - 1$ ponavljanja kroz sve lukove, budući da je svaki najkraći $s - i$ put jednostavan i koristit će najviše $n - 1$ lukova.

Ovaj se algoritam tradicionalno pripisuje zajedno Bellmanu i Fordu iako su ga otkrili i drugi otprilike u isto vrijeme. Bellman-Ford algoritam prikazujemo u Algoritmu 3. Algoritam neće raditi ispravno ako graf sadrži cikluse negativnih težina, ali mi ćemo ga prvo analizirati, a zatim vidjeti kako ga modificirati kako bi se otkrila negativna težina ciklusa.

Algorithm 3 Bellman-Ford algoritam za pronalazak najkraćeg puta

$d(i) \leftarrow \infty, \forall i \in V$
$p(i) \leftarrow null, \forall i \in V$
$d(s) \leftarrow 0$
for $k = 1$ do $n - 1$ **do**
  for all (i,j) $\in A$ **do**
    if $d(j) > d(i) + c(i, j)$ **then**
      $d(j) \leftarrow d(i) + c(i, j)$
      $p(j) \leftarrow i$
    end if
  end for
end for

Lema 1.18 *Bilo koja konačna udaljenost $d(i)$ je duljina nekog $s - i$ puta u mreži.*

Dokaz. Lema se lako dokaže indukcijom. Na početku algoritma jedina konačna udaljenost je $d(s) = 0$ što je duljina $s - s$ put. Zatim kad god ažuriramo udaljenost $d(j)$ postavljamo $d(j) = d(i) + c(i, j)$ tako da je $d(j)$ duljina $s - i$ puta duljine $d(i)$ (koja postoji po indukciji) plus duljina luka (i, j). ■

Lema 1.19 *Nakon k iteracija Bellman-Ford algoritma (Algoritam 3) svaka udaljenost $d(i)$ je najviše duljina najkraćeg $s - i$ puta koji koristi najviše k lukova.*

Dokaz. Dokazujemo da je tvrdnja istinita za bazu indukcije $k = 0$, tj. kada ne koristimo nijedan luk. U tom slučaju najkraći put P od s do j je jednostavni put od vrha s do vrha j duljine 0 što znači da je $d(j) = 0$.

Pretpostavimo da je izjava istinita za k lukova, tj. pretpostavljamo da je najkraći put P od s do j koji koristi najviše k lukova duljine $d(j)$. Moramo dokazati da iz toga slijedi da je izjava istinita i za $k+1$ lukova, tj. da je najkraći put P od s do j koji koristi najviše $k+1$ lukova takav da je $d(j)$ duljina tog puta. Prema pretpostavci indukcije podput P od s do i s najviše k lukova mora biti najkraći put od s do i koji koristi najviše k lukova. Dakle, $d(i)$ je najviše duljina tog najkraćeg puta. Nakon dodavanja luka (i, j) na taj najkraći put, dobivamo put od s do j s najviše $k+1$ lukova. Ako postoji kraći put od s do j koji koristi najviše $k+1$ lukova, onda bi se mogao sastojati od kraćeg podputa od s do i koji koristi najviše k lukova i luka (i, j). To bi značilo da bi $d(j)$ bilo manje od $d(i) + c(i, j)$ što je suprotno pretpostavci da je $d(i)$ najviše duljina najkraćeg puta od s do i koji koristi najviše k lukova. Stoga, nakon k-te iteracije vrijedi da je $d(j)$ najviše $d(i) + c(i, j)$ što znači da je $d(j)$ duljina puta P. Time je tvrdnja dokazana. ■

Lema 1.20 *Postoji jednostavan najkraći put od s do svakog $i \in V$ dohvatljiv iz s ako i samo ako ne postoje negativni ciklusi dohvatljivi iz s.*

Teorem 1.21 *Ako ne postoji ciklus negativne težine koji se može dohvatiti iz s onda algoritam Bellman Ford (Algoritam 3) ispravno određuje duljinu $d(i)$ najkraćeg puta od izvora s do svakog vrha $i \in V$ ako on postoji.*

Dokaz. Ako ne postoje ciklusi negativnih težina dohvatljivi iz s, onda po Lemi 1.20 slijedi da za svaki i najkraći $s - i$ put mora biti jednostavan i imati najviše $n - 1$ lukova. Po Lemama 1.18 i 1.19 slijedi da je $d(i)$ duljina najkraćeg $s - i$ puta. ■

Primijetimo da još nismo dokazali da roditeljski pokazivač p daje najkraći put u mreži. Ovu ćemo tvrdnju dokazati nakon što razmotrimo pitanje ciklusa negativnih težina.

Algoritam 3 ima vremensku složenost od $\mathcal{O}(mn)$. Svaki se luk točno ispituje $n-1$ puta što znači da algoritmu treba $\mathcal{O}(mn)$ vremena. Kako bismo smanjili broj operacija, možemo primijetimo da ne moramo uvijek provjeriti uvjet $d(j) > d(i) + c(i,j)$. Ako se vrijednost $d(i)$ nije promijenila u prethodnoj iteraciji algoritma onda nijedan luk (i,j) iz vrha i neće smanjiti vrijednost $d(j)$ u trenutnoj iteraciji. Kako bismo to bolje razumjeli, možemo koristiti koncept "skeniranja" koji dodajemo u Bellman-Ford algoritam. Skeniranje vrha provjerava sve izlazne lukove (i,j) iz tog vrha kako bi vidjeli je li $d(j) \geq d(i) + c(i,j)$. Ako je taj uvjet ispunjen onda se izvršava odgovarajuće ažuriranje. Detalje o ovoj metodi možete pronaći u Algoritmu 4. Algoritam 3 se zapravo implementira koristeći metodu "Scan" iz Algoritma 4. Ovo nam pomaže da bolje razumijemo algoritam i olakšava njegovu implementaciju.

Algorithm 4 Procedura Scan

for j such that $(i,j) \in A$ **do**
  if $d(j) > d(i) + c(i,j)$ **then**
    $d(j) \leftarrow d(i) + c(i,j)$
    $p(j) \leftarrow i$
  end if
end for

Algorithm 5 Bellman-Ford algoritam koristeći proceduru Scan

$d(i) \leftarrow \infty, \forall i \in V$
$p(i) \leftarrow null, \forall i \in V$
$d(s) \leftarrow 0$
for $k = 1$ to $n - 1$ **do**
  for all $i \in V$ **do**
    Scan(i)
  end for
end for

Sada primjećujemo da trebamo skenirati samo vrh i ako je njegova oznaka udaljenosti $d(i)$ smanjena u prethodnoj iteraciji. Ako je $d(i)$ bio nepromijenjen u prethodnoj iteraciji onda će za sve lukove (i, j) koji izlaze iz i $d(j)$ ostati najviše $d(i) + c(i, j)$.

Da bismo implementirali ovu ideju koristimo podatkovnu strukturu *queue*. *Queue* je uređena lista stavki i sadrži sljedeće operacije: *new queue*() koja vraća prazan red, $q.add(i)$ koja dodaje stavku i na kraj reda q, $q.remove()$ koja uklanja stavku s početka reda q i vraća je (ako postoji), $q.empty?$ koja provjerava je li red neprazan i $q.contains(i)?$ koja provjerava je li red q već sadrži stavku i. Pretpostavljamo da je $q.contains?$ složenosti $\mathcal{O}(1)$ umjesto $\mathcal{O}(n)$ koje bi bilo potrebno za provjeriti članstvo *skeniranjem*. Možemo implementirati operaciju na ovaj način koristeći niz. Na primjer, kada unaprijed znamo, kao što je u ovom slučaju, koje elemente red može sadržavati. Zatim možemo ponovno napisati proceduru *skeniranja* i Bellman-Ford algoritam kao što je prikazano u proceduri $QScan$ 6 i Algoritmu 7. Stavljamo vrh j u red kada se njegova oznaka udaljenosti promijenila tijekom *skeniranja*. Ako vrh j nije u redu onda njegova oznaka udaljenosti ostaje nepromijenjena i ne moramo ga skenirati.

Algorithm 6 QScan

```
for j such that (i, j) ∈ A do
    if d(j) > d(i) + c(i, j) then
        d(j) ← d(i) + c(i, j)
        p(j) ← i
        if not q.contains?(j) then
            q.add(j)
        end if
    end if
end for
```

Algorithm 7 Bellman-Ford algoritam koristeći queues

```
d(i) ← ∞, ∀i ∈ V
p(i) ← null, ∀i ∈ V
d(s) ← 0
q ← new queue()
q.add(s)
while not q.empty? do
    QScan(q.remove(),q);
end while
```

Možemo dokazati da algoritam radi ispravno sličnim induktivnim argumentom kao u dokazu Teorema 1.21 u kojem indukciju na iteracijama zamjenjujemo s indukcijom na prolazima preko reda.

Teorem 1.22 *Ako nema ciklusa s negativnom težinom dohvatljivih iz s onda Algoritam 7 ispravno određuje duljinu $d(i)$ najkraćeg puta od izvora s do svakog vrha $i \in V$ ako taj vrh postoji.*

Dokaz. Kao što je gore predloženo primjenjujemo indukciju na prolaze preko reda. Prolaz 0 završava nakon što je s inicijalno dodan u red, prolaz 1 završava nakon početnog *skeniranja* s, i općenito prolaz k završava nakon *skeniranja* svih vrhova dodanih u red čekanja u prolazu $k-1$. Pretpostavka indukcije je da je na kraju k-tog prolaza $d(i)$ najviše duljina najkraćeg $s-i$ puta s najviše k lukova, nastavak dokaza je analogan dokazu Teorema 1.21. Ako nema ciklusa s negativnom težinom do kojih se može doći iz s onda najkraći $s-i$ put može sadržavati najviše $n-1$ lukova i stoga će na kraju $(n-1)$-og prolaza vrijednost $d(i)$ biti duljina ovog puta. Također, budući da je $d(i)$ duljina najkraćeg puta od s do i vrijedi: $d(j) \leq d(i) + c(i,j)$ za sve $(i,j) \in A$ koji su dohvatljivi iz s jer bi inače postojao kraći put od s do j. Dakle, nema dodatnih vrhova koji će biti dodani u red i algoritam završava. ■

Dakle, ima najviše $n-1$ prolaza, a svaki prolaz razmatra svaki vrh najviše jednom, najviše m lukova se razmatra u svakom prolazu. Stoga je složenost algoritma $\mathcal{O}(mn)$.

1.3.1 Detekcija ciklusa negativnih težina

Sada želimo modificirati Bellman-Ford algoritam tako da završava s najkraćim putem ako nema ciklusa s negativnom težinom dohvatljivih iz s, ili se zaustavlja ako detektira ciklus negativne težine dohvatljiv iz s u ulaznom grafu. Za početak naše rasprave, vraćamo se našoj početnoj verziji Bellman-Ford Algoritma 3. Prikazujemo uvjet koji možemo provjeriti na kraju algoritma koji omogućuje da se uvjerimo da ne postoji ciklus negativne težine.

Lema 1.23 *Ne postoje ciklusi negativne težine koji se mogu postići iz s ako i samo ako je $d(j) \leq d(i) + c(i,j)$ za sve $(i,j) \in A$ gdje je i dohvatljiv iz s*

na kraju Algoritma 3.

Dokaz. Ako na kraju Algoritma 3 vrijedi $d(j) > d(i) + c(i,j)$ za neki $(i,j) \in A$ onda ovo implicira da bismo mogli pronaći kraći put do j uzimajući put $s - i$ duljine $d(i)$ (koji postoji prema Lemi 1.18) nego luk (i,j) stoga algoritam nije točno izračunao najkraći $s - j$ put. Budući da Teorem 1.22 tvrdi da algoritam izračunava najkraće duljine puta ako ne postoji ciklus negativne težine koji se može dohvatiti iz s pretpostavka mora biti netočna i mora postojati ciklus negativne težine koji se može postići iz s. Ako za sve $(i,j) \in A$ s i dohvatljivim iz s vrijedi $d(j) \leq d(i) + c(i,j)$ onda je bilo koji ciklus C dohvatljiv iz s. Iz tog slijedi:

$$\sum_{(i,j)\in C} c(i,j) \geq \sum_{(i,j)\in C} (d(j) - d(i)) = 0. \tag{1.1}$$

Stoga C ne može imati negativnu težinu. ■

Lema 1.23 daje neposrednu metodu za otkrivanje postoji li negativna težina ciklusa iako ne kaže kako pronaći ciklus jednostavno provjeravamo na kraju algoritma je li $d(j) \leq d(i) + c(i,j)$ za sve $(i,j) \in A$. Ako je uvjet ispunjen onda nema ciklusa negativnih težina, inače ima. Ovaj algoritam prikazujemo u Algoritmu 8.

Algorithm 8 Algoritam za detekciju negativnog ciklusa

```
d(i) ← ∞, ∀i ∈ V
p(i) ← NULL, ∀i ∈ V
d(s) ← 0
for k = 1 do n − 1 do
    for all (i, j) ∈ A do
        if d(j) > d(i) + c(i, j) then
            d(j) ← d(i) + c(i, j)
            p(j) ← i
        end if
    end for
end for
for all (i, j) ∈ A do
    if d(j) > d(i) + c(i, j) then
        return "negative-cost cycle exists"
    end if
end for
```

Nedostatak ovog algoritma, osim što ne pronalazi ciklus, je to što je potrebno n iteracija i $\mathcal{O}(mn)$ vremena. Željeli bismo da algoritam završi ranije ako je moguće tj. kada se otkrije ciklus negativnih težina. Da bismo dali takav algoritam vratimo se na roditeljske pokazivače koje koristi algoritam i uvedimo koncept roditeljski graf koji označavamo s G_p. Roditeljski graf sastoji se od skupa lukova $(p(j), j)$ za sve $j \in V$ za koje je $p(j)$ definiran. Zamislimo da pratimo roditeljski pokazivač unatrag od vrha j do početnog vrha s kako bismo dobili put od s do j u grafu. U tom procesu roditeljski graf G_p će nam pružiti usmjeren put od s prema j koji sadrži samo neke od lukova iz izvornog grafa G. Sada ćemo pokazati da će se ciklus negativne težine pojaviti

u roditeljskom grafu G_p ako i samo ako postoji ciklus negativne težine koji je dohvatljiv iz početnog vrha s. Ovo nam sugerira jednostavan algoritam za pronalaženje ciklusa negativne težine: u svakom koraku provjeravamo roditeljski graf G_p kako bismo detektirali prisutnost ciklusa. Ako pronađemo ciklus negativne težine u G_p onda znamo da postoji ciklus negativne težine koji je dohvatljiv iz početnog vrha s. Na taj način roditeljski pokazivač nam omogućava da pratimo put unatrag određenog vrha do početnog vrha te nam pomaže u pronalaženju ciklusa negativne težine u grafu.

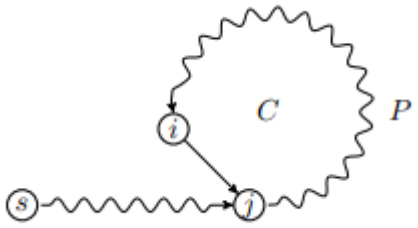
Lema 1.24 *U bilo kojem trenutku u Algoritmu 3, ako je luk (i, j) u roditeljskom grafu G_p onda je $d(j) \geq d(i) + c(i, j)$.*

Lema 1.25 *Za bilo koji $h - l$ put P u roditeljskom grafu G_p težina puta $\sum_{(i,j)\in P} c(i, j)$ je najviše $d(l) - d(h)$.*

Dokaz. Prema Lemi 1.24 vrijedi: $\sum_{(i,j)\in P} c(i, j) \leq \sum_{(i,j)\in P}(d(j) - d(i)) = d(l) - d(h)$ ■

Lema 1.26 *Pretpostavimo da postoji ciklus u G_p. Prvi ciklus koji se pojavio u G_p ima negativnu težinu i dohvatljiv je iz s.*

Dokaz. Ciklus C se morao pojaviti u roditeljskom grafu G_p jer smo dodali luk (i, j) u G_p, a j je bio dio $s - i$ staze u G_p prije ažuriranje (vidi Sliku 1.8). Ako je P put u G_p od j do i i $d(i)$ i $d(j)$ su udaljenosti prije ažuriranja onda prema gore navedenom i Lemi 1.24 imamo $\sum_{(k,l)\in C} c(k, l) = c(i, j) + \sum_{(k,l)\in P} c(k, l) \leq c(i, j) + d(i) - d(j)$. Budući da mi izvršimo ažuriranje, to mora biti zato što je $d(j) > d(i) + c(i, j)$ i stoga $c(i, j) + d(i) - d(j) < 0$ i ukupna težina lukova u ciklusu C je negativna. ■



Slika 1.8: Ilustracija dokaza Leme 1.26. Put P, označen valovitom linijom, je prije ažuriranja koje dodaje luk (i, j) roditeljskom grafu.

Dakle, ako pronađemo ciklus u roditeljskom grafu G_p onda Lema 1.26 pokazuje da on mora biti ciklus negativne težine. Kao prvi pokušaj algoritma koji koristi ovaj uvid modificiramo Bellman-Ford algoritam da poduzima n iteracija, ali pri svakom ažuriranju provjerimo postojanje ciklusa u G_p. Dajemo algoritam u Algoritmu 9. Ovaj algoritam nije brz jer treba $\mathcal{O}(n)$ vremena za provjeru $G_p - a$ za ciklus dajući ukupno vrijeme $\mathcal{O}(mn^2)$. Međutim, kada dokažemo da je ovaj algoritam ispravan, pokazat ćemo kako ga možemo modificirati tako da se vrijeme za provjeru ciklusa u G_p može smanjiti tijekom operacija koje se koriste za ažuriranje roditeljskog grafa.

Algorithm 9 Algoritam za detekciju negativnog ciklusa

```
d(i) ← ∞, ∀i ∈ V
p(i) ← null, ∀i ∈ V
d(s) ← 0
for k = 1 do n do
    for all (i, j) ∈ A do
        if d(j) > d(i) + c(i, j) then
            d(j) ← d(i) + c(i, j)
            p(j) ← i
            if G_p has a cycle C then then
                return C
            end if
        end if
    end for
end for
```

Lema 1.27 *Ako Algoritam 9 ne vrati ciklus onda je* $d(s) = 0$.

Dokaz. Ako se $d(s)$ ikada promijeni onda se njegov roditeljski pokazivač $p(s)$ mijenja u vrh i tako da postoji neki $s - i$ put, a to rezultira ciklusom u G_p. Dakle, ako nema ciklusa onda je $d(s) = 0$ na kraju algoritma. ■

Lema 1.28 *Tijekom izvođenja Algoritma 9 roditeljski graf* G_p *je stablo usmjereno iz* s.

Dokaz. Dokaz ćemo provesti indukcijom. Pokazujemo da lukovi $(p(j), j)$ za sve j takve da je $p(j) \neq null$ imaju oblik stabla usmjerenog iz s. U početku nema lukova u G_p. Ako dodamo luk (i, j) u G_p to je zato što je $d(i)$ konačan, a indukcijom je vrh i dohvatljiv iz s u roditeljskom grafu G_p. Ako dodavanje

luka (i, j) u G_p uzrokuje ciklus onda algoritam završava, inače ažuriramo $p(j)$ na i i imamo put od s do j. ■

Teorem 1.29 *Algoritam 9 ili ispravno izračunava najkraći put od s do svih vrhovova $i \in V$ ili ispravno detektira i vraća dohvatljiv ciklus negativne težine iz vrha s.*

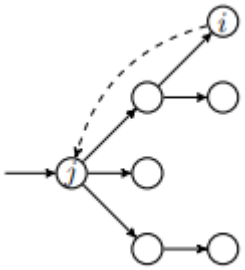
Dokaz. Prema Lemi 1.26 ako algoritam vraća ciklus C onda je C ciklus negativne težine. Sada pokazujemo da ako postoji ciklus negativne težine C koji se može postići iz s algoritam će vratiti taj ciklus. Prema Lemi 1.23 ako postoji ciklus negativne težine dohvatljiv iz s onda na kraju algoritma, u n-toj iteraciji, mora postojati neki $(i, j) \in A$ takav da je $d(j) > d(i) + c(i, j)$ za i dohvatljiv iz s. Zatim prema Lemi 1.19 nakon ažuriranja $d(j)$ u n-toj iteraciji $d(j)$ je manji od težine bilo kojeg jednostavnog $s - j$ puta budući da nakon $n - 1$ ponavljanja nije bilo veće težine od bilo kojeg jednostavnog puta i od tada se smanjila. Međutim, ako nema ciklusa u G_p onda prema Lemama 1.25 i 1.28 mora postojati jednostavan $s - j$ put P u G_p najveće težine $d(j) - d(s)$. Prema Lemi 1.27 je $d(s) = 0$. Stoga je težina jednostavnog puta P najviše $d(j)$, ali to je kontradikcija jer $d(j)$ ima manju težinu od bilo kojeg jednostavnog $s - j$ puta. Dakle, ako je $d(j) > d(i) + c(i, j)$ za neki (i, j) u n-toj iteraciji onda ciklus mora postojati u G_p. ■

Lema 1.30 *Postoji jednostavan najkraći put iz s do svakog vrha $i \in V$ dohvatljivog iz s ako i samo ako nema ciklusa negativne težine dohvatljivih iz s.*

Korolar 1.31 *Ako ne postoji ciklus negativne težine koji se može dohvatiti iz s na grafu onda Algoritmi 3, 5 i 7. ispravno pronalaze najkraći put od s do svih vrhova $i \in V$ u grafu G_p.*

Dokaz. Ako ne postoji ciklus negativne težine koji se može dohvatiti iz s u grafu onda je $d(i)$ duljina najkraćeg $s-i$ puta u grafu (prema Teoremima 1.21 i 1.22) postoji jednostavan $s-i$ put koji je najkraći (prema Lemi 1.30) također postoji jednostavan $s-i$ put u G_p duljine najviše $d(i)$ (prema Lemama 1.25 i 1.27). Dakle, $s-i$ put u G_p mora biti najkraći $s-i$ put. ■

Na kraju, razmatramo kako možemo imati prednost algoritma koji završava ranije ako detektira ciklus u G_p u $\mathcal{O}(mn)$ vremena. Da bismo to učinili, modificiramo Algoritam 7 koji koristi redove i operaciju skeniranja. Primjećujemo da postavljanje $p(j)$ na i uzrokuje ciklus u roditeljskom grafu G_p točno onda kada je i u podstablu s korijenom u j pa možemo provjeriti uzrokuje li postavljanje $p(j)$ na i ciklus počevši od j i istražujući podstablo ukorijenjeno u j praćenjem lukova u roditeljskom grafu. Ako nađemo i onda postavljamo $p(j)$ na i i to će uzrokovat ciklus (vidi Sliku 1.9). Ova operacija ima složenost $\mathcal{O}(n)$ i tako se čini da nismo mnogo napredovali. Međutim, ako bismo svaki luk prešli najviše jednom vrijeme potrebno da se obiđe svaki luk u podstablu ukorijenjenom u j računa se prema vremenu u kojem je luk nastao. Kako bismo bili sigurni da ne prelazimo isti luk više puta, ako nema ciklusa, tj. ako ne nalazimo i u podstablu od j onda brišemo svaki luk u podstablu od j. Posebno, za svaki i u podstablu koje ima korijen u j postavljamo $p(i')$ na *null*. Ovaj algoritam sažimamo u Algoritmu 11.



Slika 1.9: Ilustracija otkrivanja ciklusa putem podstabala

Algorithm 10 NCCScan

```
for j such that (i, j) ∈ A do
    if d(j) > d(i) + c(i, j) then
        Traverse subtree of G_p rooted in j
        if i in subtree of G_p rooted at j then
            Let C be the cycle closed by (i,j) and j-i path in G_p
            return C
        else if  then
            for all i' in subtree of G_p rooted at j do
                p(i') ← null
            end for
        end if
    end if
    d(j) ← d(i) + c(i, j)
    p(j) ← i
    if not q.contains?(j) then
        q.add(j)
    end if
end for
```

Algorithm 11 Konačni algoritam za detekciju ciklusa negativnih težina

```
d(i) ← ∞, ∀i ∈ V
p(i) ← null, ∀i ∈ V
d(s) ← 0
q ← new queue()
NCCScan(s,q)
while not q.empty? do
    i ← q.remove()
    if p(i) ≠ null then
        NCCScan(i,q)
    end if
end while
```

Sada dokazujemo da je algoritam ispravan i da mu je složenost $(\mathcal{O}mn)$. Budući da smo uklonili bridove roditeljskog grafa i nismo *skenirali* svaki ažurirani vrh u svakom prolazu kroz red trebat ćemo modificirati neke pomoćne leme i sveukupni pristup dokazivanju. Kao i u dokazu Teorema 1.22, primijenit ćemo indukciju po prolascima algoritma kroz red.

Lema 1.32 *Tijekom izvođenja Algoritma 11, roditeljski graf G_p je stablo usmjereno iz s.*

Dokaz. Dokaz provodimo indukcijom. Pokazujemo da svi lukovi $(p(j), j)$ za svaki vrh j takav da je $p(j) \neq null$ imaju oblik stabla usmjerenog iz s. Inicijalno nema lukova u G_p. Ako dodamo luk (i, j) u graf G_p to je zato što smo skenirali i, a to možemo učiniti samo ako je $p(i) \neq null$ stoga je indukcijom i dohvatljiv iz s u roditeljskom grafu G_p. Ako dodavanje luka (i, j) uzrokuje ciklus onda se algoritam prekida, inače ažuriramo $p(j)$ na i i

uklonimo cijelo podstablo kojem je korijen j tako da ono što ostaje još uvijek bude stablo, ali s korijenom s. ■

Lema 1.33 *Ako Algoritam 11 ne vrati ciklus onda je* $d(s) = 0$.

Dokaz. Ako se $d(s)$ ikada promijeni onda je to zbog razmatranja luka (i, s) tijekom skeniranja vrha i. Možemo skenirati i samo ako je $p(i)$ definiran pa je prema Lemi 1.32 i dohvatljiv iz s u grafu G_p. Stoga je i u podstablu s korijenom u s i algoritam će vratiti negativan ciklus i završiti. ■

Lema 1.34 *U bilo kojem trenutku u Algoritmu 11 za bilo koji luk* (i, j) *u roditeljskom grafu* G_p *vrijedi:* $d(j) = d(i) + c(i, j)$.

Dokaz. U slučaju da se luk (i, j) dodaje u graf G_p vrijedi $d(j) = d(i) + c(i, j)$. Ako se $d(i)$ ažurira putem *skeniranja* vrha h s lukom (h, i) i ne vraća negativni ciklus, budući da je j u podstablu od i u G_p, onda se luk (i, j) briše u grafu G_p. ■

Lema 1.35 *Tijekom izvođenja Algoritma 11 bilo koji* l *u roditeljskom grafu* G_p *ima* $d(l)$ *jednak težini puta* $s - l$ *u grafu* G_p.

Dokaz. Slijedeći dokaz Leme 1.25 i koristeći Leme 1.33 i 1.34 za neki $s - l$ put P u grafu G_p vrijedi:

$$\sum_{(i,j)\in P} c(i, j) = \sum_{(i,j)\in P} (d(j) - d(i)) = d(l) - d(s) = d(l).$$

■

Dokaz Teorema 1.22 proveli smo indukcijom po broju prolaza da bismo dokazali da je nakon k prolaza $d(i)$ uvijek jednak najviše duljini najkraćeg $s-i$ puta s najviše k lukova. Budući da možemo preskočiti *skeniranje* vrha i kada je $p(i) = null$, ne možemo koristiti istu indukciju. Međutim možemo dokazati sljedeće:

Lema 1.36 *Svaki $d(i)$ ažuriran u k-tom prolazu je duljina jednostavnog $s-i$ put s najmanje k lukova.*

Dokaz. Kad god se udaljenost $d(i)$ ažurira i postaje dio stabla u grafu G_p prema Lemi 1.35 je $d(i)$ duljina jednostavnog $s-i$ puta u stablu. Na kraj nultog prolaza $s-s$ put je u stablu i $d(s)$ je duljina ovog puta koji ima nula lukova. Ako se udaljenost $d(j)$ ažurira u k-tom prolazu uzimajući u obzir luk (i,j) onda je $d(i)$ ažuriran u $k-1$ prolazu, a indukcijom se pokaže da $s-i$ put duljine $d(i)$ ima najmanje $k-1$ lukova u sebi. Prema tome $s-j$ put formiran od $s-i$ puta plus luk (i,j) ima najmanje k lukova u sebi. ■

Budući da sada možemo imati vrhove i s konačnom udaljenošću $d(i)$ koji ipak nemaju put od s do i u G_p kroz algoritam, moramo osigurati da postoji put od s do i u konačnom roditeljskom grafu G_p. Sljedeća lema dokazuje da je to doista tako.

Lema 1.37 *Pretpostavimo da ne postoje negativni ciklusi dohvatljivi iz s. Ako se $d(j)$ ažurira na duljinu najkraćeg $s-j$ puta onda se $d(j)$ ne ažurira u budućim prolazima i j je dio stabla u G_p kroz ostatak algoritma.*

Dokaz. Budući da nema ciklusa s negativnom težinom do kojih se može doći iz s, $s-s$ put duljine 0 je najkraći $s-s$ put. Inicijalno, $d(s)$ je postavljen na 0 i s je dio stabla u G_p kroz cijeli algoritam. Sada razmotrimo najkraći $s-j$ put za $j \neq s$. Neka je (i,j) završni luk puta. Prema pretpostavci u nekom trenutku $d(j)$ se ažurira na duljina najkraćeg $s-j$ puta. Primijetimo da kada se $d(j)$ ažurira na duljinu najkraćeg $s-j$ put svaki vrh l na putu već ima $d(l)$ jednaku duljini najkraćeg $s-l$ puta. Primijetimo da $d(l)$ neće biti ažuriran u budućim iteracijama, prema Lemi 1.36, $d(l)$ je uvijek duljina nekog jednostavnog $s-l$ puta i ne postoji kraći jednostavni $s-l$ put. Kada se $d(j)$ ažurira na duljinu najkraćeg $s-j$ puta, luk (i,j) se dodaje stablu u

G_p. Luk (i, j) se nikada ne uklanja jer nijedan od prethodnika od j na putu se ne ažurira. ■

Konačno, možemo pokazati da je algoritam ispravan.

Teorem 1.38 *Algoritam 11 se izvršava u $\mathcal{O}(mn)$ vremenu i ispravno računa najkraći put od s do svakog vrha $i \in V$ ili ispravno detektira i vraća ciklus negativne težine dohvatljiv iz s.*

Dokaz. Ako algoritam ažurira udaljenosti $d(i)$ u n-tom prolazu algoritma onda je po Lemi 1.36 to duljina jednostavnog $s - i$ puta od najmanje n lukova što je kontradikcija i ukazuje na prisutnost ciklusa negativnih težina. Stoga se algoritam završava do kraja n-tog prolaza bilo vraćanjem ciklusa negativne težine ili ne. Ako algoritam vrati ciklus C onda je prema Lemi 1.26 on negativne težine i dohvatljiv je iz s. Budući da algoritam mora završiti nakon n-tog prolaza, a u najgorem slučaju svaki luk se obrađuje u svakom prolazu, vrijeme rada je $\mathcal{O}(mn)$ osim za operaciju obilaska podstabala koja se koristi u otkrivanju ciklusa. Međutim, mi računamo težinu prelaska svakog luka u podstablu do operacije koja je dodala luk u roditeljski graf na prvom mjestu. Kako se luk uklanja nakon prelaska podstabla, jasno je da ne računamo događaj dodavanja luka više od jednom. Tako se algoritam izvodi za $\mathcal{O}(mn)$ vremena. Ako algoritam završi bez vraćanja ciklusa onda nisu potrebna daljnja ažuriranja udaljenosti tako da je $d(j) \leq d(i) + c(i, j)$ za svaki $(i, j) \in A$ s i dohvatljivim iz s pa iz Leme 1.23 slijedi da algoritam ispravno tvrdi da nema ciklusa negativnih težina koji se mogu postići iz s. Neka je P najkraći $s - l$ put. Tada su svi vrhovi iz P dohvatljivi iz s tako da je $\sum_{(i,j)\in P} c(i, j) \geq \sum_{(i,j)\in P}(d(j) - d(i)) = d(l) - d(s)$. Po Lemi 1.27 slijedi da je $d(s) = 0$ tako da je $d(l) \leq \sum_{(i,j)\in P} c(i, j)$. Budući da je, prema Lemi 1.36, $d(l)$ uvijek duljina jednostavnog $s - l$ puta, ne može biti slučaj da je $d(l)$ manji od težine puta P pa mora biti jednak težini puta P. Tada prema

Lemi 1.37 vrh l mora biti dio stabla u G_p, a prema Lemi 1.35 $d(l)$ je jednak duljini puta u G_p. ■

Iako se ovaj konačni algoritam čini kompliciranijim od prethodnih računalne studije su pokazale da je učinkovitiji za pronalazak najkraćeg puta i otkrivanje ciklusa negativnih težina.

Poglavlje 2

Mrežni protok

Sada se okrećemo osnovnom problemu teorije mrežnog protoka, problemu maksimalnog protoka, i njegovom dualnom problemu, problemu minimalnog $s-t$ reza. Ova dva problema pokazala su se izuzetno korisnima u modeliranju različitih problema koji uključuju mreže svih vrsta: cestovne mreže, željezničke mreže, računalne mreže, društvene mreže i druge. Obično modeliramo protok materijala s jednog dijela mreže na drugi, a protok može biti u obliku automobila, vlakova, preporuka, bitova, povjerenja i mnogih drugih predmeta i pojmova. Zanimljivo je da su se ova dva problema također pokazala korisnima u modeliranju problema u kojima nije očito da je uključena mreže ili protok materijala. Navest ćemo nekoliko primjera: jedan o odlučivanju o pravednom načinu raspodjele vožnji u zajedničkom prijevozu, drugi u određivanju koje bejzbolske momčadi ne mogu osvojiti svoju diviziju i treći u pronalaženju najgušćeg podgrafa u zadanom neusmjerenom grafu.

Definicija 2.1 *Mreža je usmjereni graf $G = (V, A)$ u kojem svaki brid $(i, j) \in A$ ima nenegativan kapacitet $u(i, j) \geq 0$. Za $(i, j) \notin A$ definiramo $u(i, j) = 0$. Mreža sadrži dva posebno naznačena vrha, izvor s i ponor t.*

Ideja je pronaći najveći ukupni protok tvari od izvora do ponora. Pretpos-

tavimo da ne postoji brid koji ulazi u s, kao ni brid koji izlazi iz t. Takvu mrežu ponekad nazivamo $s-t$ mreža. Osim toga pretpostavimo da se svaki vrh grafa nalazi na nekom od puteva od izvora s do ponora t.

Definicija 2.2 *Za danu $s-t$ mrežu protok (ili $s-t$ protok) definiramo kao funkciju f koja preslikava svaki brid u nenegativan realan broj i zadovoljava sljedeće uvjete:*

1. *ograničenje kapaciteta:*
   Za svaki brid $(i,j) \in A$ vrijedi $0 \leq f(i,j) \leq u(i,j)$.

2. *očuvanje protoka:*
   Za sve $i \in V \backslash \{s,t\}$ suma protoka kroz bridove koji ulaze u vrh i jednaka je sumi protoka kroz bridove koji izlaze iz vrha i tj.

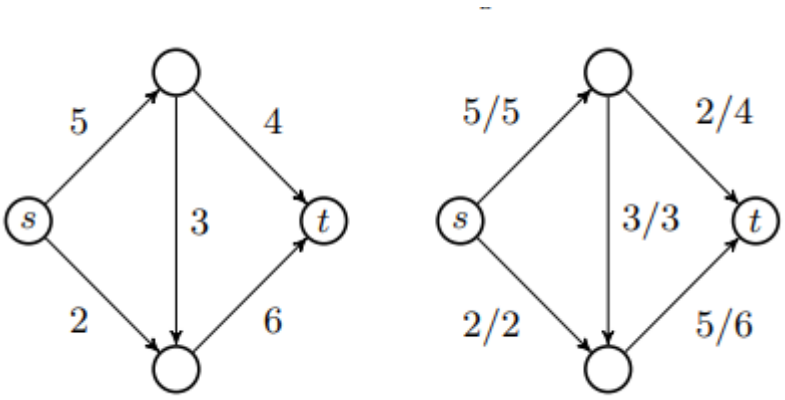
$$\sum_{k:(k,i)\in A} f(k,i) = \sum_{k:(i,k)\in A} f(i,k).$$

Svaki protok f povezujemo s vrijednošću koju označavamo s $|f|$.

Definicija 2.3 *Vrijednost $s-t$ protoka f je definirana kao:*

$$|f| = \sum_{k:(s,k)\in A} f(s,k) - \sum_{k:(k,s)\in A} f(k,s).$$

Cilj problema maksimalnog protoka je pronaći $s-t$ protok f koji maksimizira $|f|$. Kažemo da je protok maksimalan (ili optimalan) ako je protok najveće vrijednosti. Napomenimo da je zbog nenegativnih cjelobrojnih kapaciteta protok $f \geq 0$ jer je $f(i,j) \geq 0$ za sve $(i,j) \in A$ pa je i vrijednost maksimalnog protoka uvijek nenegativna. Budući da su kapaciteti nenegativni cijeli brojevi i protok je uvijek nenegativan. Na Slici 2.1 dajemo primjer problema s maksimalnim protokom i protokom.



Slika 2.1: Broj pored svakog luka na slici lijevo predstavlja kapacitet luka. Oznaka "x/y" predstavlja omjer protoka i kapaciteta na svakom bridu.

Nadalje, razmotrimo samo gornje granice protoka, a ne i donje granice. Za svaki luk (i, j) u skupu lukova A uvest ćemo dodatni obrnuti luk (j, i) u skup A. Ako je $f(i, j)$ protok na luku (i, j) onda je $-f(i, j)$ protok na luku (j, i). Ovo svojstvo se naziva simetrija iskrivljenja. Sada donja granica $f(i, j) \geq 0$ može biti zamijenjena s $u(j, i) = 0$, što znači da ako je $f(j, i) \leq u(j, i)$ onda je $f(i, j) = -f(j, i) \geq -u(i, j) = 0$. Osim toga, prema novoj definiciji, zbroj protoka na izlaznim granama vrha i jednak je protoku iz tog vrha. Drugim riječima, ako je A' novi skup lukova koji uključuje i obrnute lukove, tada vrijedi:

$$\sum_{k:(i,k)\in A'} f(i, k) = \sum_{k:(i,k)\in A} f(i, k) - \sum_{k:(k,i)\in A} f(k, i).$$

Iz toga slijedi:

$$\sum_{k:(i,k)\in A'} f(i, k) = 0$$

za $i \neq s, t$. Na Slici 2.2 je vidljiva razlika između starog i novog shvaćanja očuvanja mrežnog protoka. Sukladno tom, vrijednost protoka je

$$|f| = \sum_{k:(s,k)\in A'} f(s, k).$$

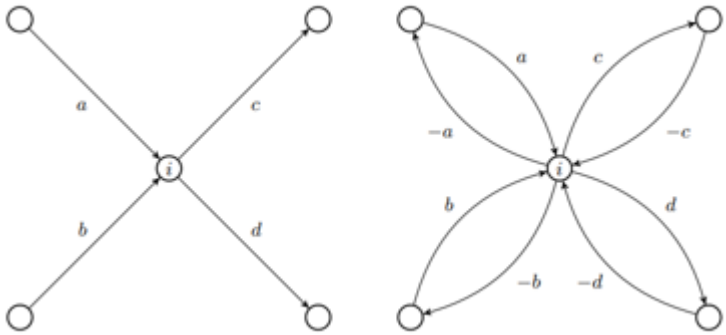
Od sada ćemo jednostavno novi skup bridova označavati s A.

Definicija 2.4 *Za danu $s-t$ mrežu, protok definiramo kao funkciju $f : A \to \mathbb{R}$ koja zadovoljava sljedeća tri svojstva:*

1. *za sve $(i, j) \in A$*
$$f(i, j) \leq u(i, j);$$

2. *za sve $i \in V$ takve da je $i \neq s, t$*
$$\sum_{(i,k)\in A} f(i, k) = 0;$$

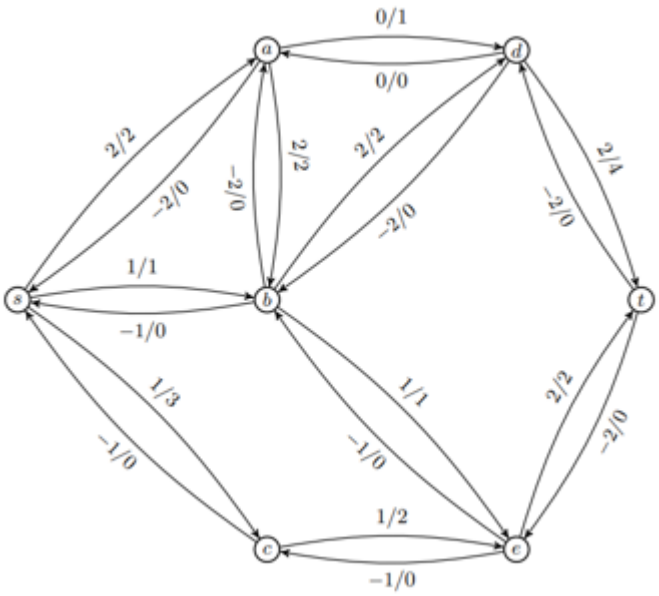
3. *za sve $(i, j) \in A$*
$$f(i, j) = -f(j, i).$$



Slika 2.2: Očuvanje protoka kroz dvije različite definicije. S lijeve strane imamo protok prema Definiciji 2.2,a s desne prema Definiciji 2.4. S obzirom na simetriju iskrivljenja dovoljno je zbrojiti izlazne lukove kako bi se provjerilo je li protok nula. Konkretno, s lijeve strane imamo $f^{out} - f^{in}$ što je $c + d - (a + b)$, a s desne strane suma na izlaznim bridovima je $(-a) + (-b) + c + d$.

2.1 Ford-Fulkerson algoritam

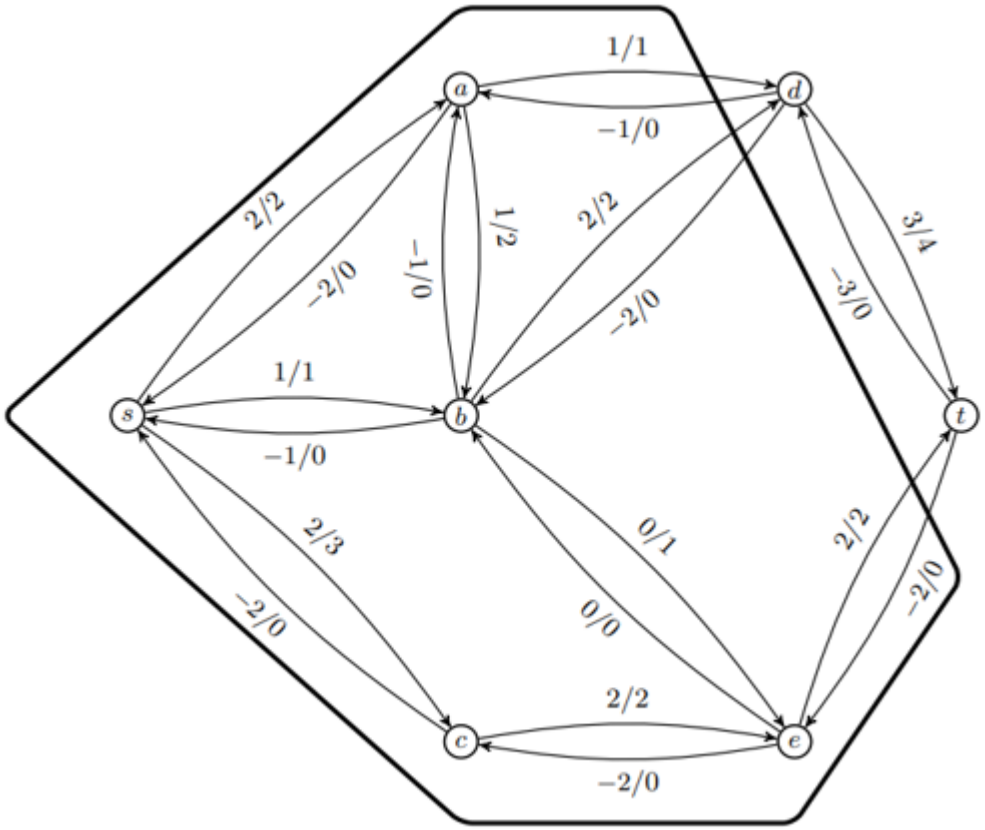
Sada započinjemo našu raspravu o problemu maksimalnog mrežnog protoka pitajući se kako možemo utvrditi je li određeni protok maksimalan ili ne. Razmotrimo protok prikazan na Slici 2.3 Je li taj protok maksimalan? Trebali bismo provjeriti da za bilo koji put od s do t koji ima bridove s pozitivnim kapacitetom ne možemo uvećati protok duž tog puta zadovoljavajući ograničenje kapaciteta.



Slika 2.3: Primjer protoka vrijednosti $|f| = 4$.

Zapravo, protok prikazan na Slici 2.3 nije maksimalan. Možemo dobiti veći protok vrijednosti $|f| = 5$ kao što je prikazano na Slici 2.4. Također, vidimo da bilo koja jedinica protoka od s do t mora proći kroz jedan od lukova iznutra iz područja u dijagramu prema van iz tog područja stoga tvrdimo da vrijednost protoka ne može biti veća od ukupnog kapaciteta tih lukova koji je $u(a, d) + u(b, d) + u(e, t) = 1 + 2 + 2 = 5$. Ako je ova tvrdnja točna onda

je tok vrijednosti 5 maksimalan.



Slika 2.4: Drugi protok vrijednost $|f| = 5$. Zatvorene linije oko grafa prikazuju da protok ne može biti veći zato što je 5 jedinica kapaciteta dostupno na lukovima koji napuštaju zatvoreni skup vrhova.

Sada želimo formalizirati intuiciju koju smo koristili ranije. Kažemo da je $s-t$ rez skup vrhova $S \subseteq V$ takav da je $s \in S$ i $t \notin S$ i to odgovara području na Slici 2.4. Označavamo skup svih lukova koji napuštaju skup S s $\delta^+(S)$, tj. $\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}$. Ponekad kažemo da su lukovi u $\delta^+(S)$ lukovi u rezu definiranom pomoću skupa S. Skup svih lukova koji ulaze u skup S označavamo s $\delta^-(S) = \{(i,j) \in A : i \notin S, j \in S\}$. Kapacitet $s-t$

reza S je zbroj kapaciteta svih lukova koji napuštaju S (ili se nalaze u rezu definiranom pomoću S) i zapisujemo ga kao $u(\delta^+(S)) = \sum_{(i,j)\in\delta^+(S)} u(i,j)$.

Lema 2.5 *Za bilo koji $s-t$ protok f i bilo koji $s-t$ rez S je $|f| \leq u(\delta^+(S)$.*

Korolar 2.6 *Neka je f $s-t$ protok i S $s-t$ rez. Tada je $f(i,j) = u(i,j)$ za sve $(i,j) \in \delta^+(S)$ ako i samo ako je $|f| = u(\delta^+(S))$.*

Dokaz. Ako je $f(i,j) = u(i,j)$ za sve $(i,j) \in \delta^+(S)$ onda je tvrdnja dokazana. Ako je $|f| = u(\delta^+(S))$ onda mora biti $f(i,j) = u(i,j)$ za sve $(i,j) \in \delta^+(S)$. ■

Primijetimo da za tok na Slici 2.4 vrijedi $f(a,d) = u(a,d)$, $f(b,d) = u(b,d)$ i $f(e,t) = u(e,t)$ za tri luka u $\delta^+(S)$ koji pripadaju $s-t$ rezu S.

Definicija 2.7 *Kažemo da je S^* minimalni $s-t$ rez ako ima najmanji kapacitet*

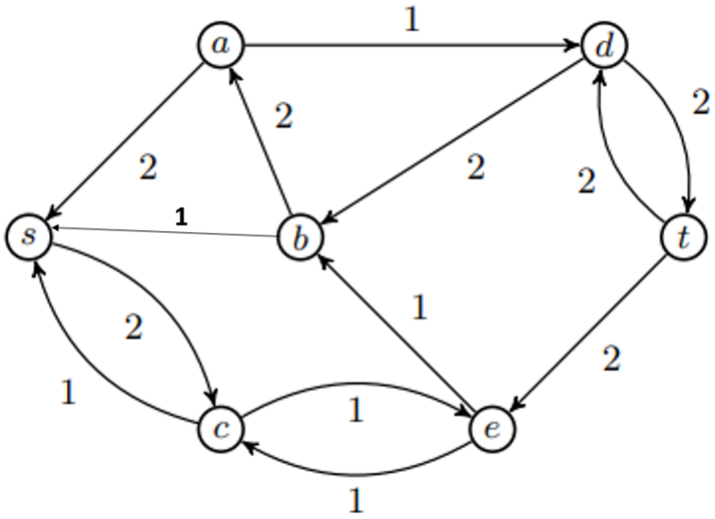
$$u(\delta^+(S)^*) = \min_{S\subseteq V, s\in S, t\notin S} u(\delta^+(S)).$$

Prema Lemi 2.5, vrijednost maksimalnog protoka je najviše jednaka kapacitetu minimalnog $s-t$ reza. U našem primjeru na Slici 2.4 vidjeli smo da ove dvije veličine mogu biti jednake. Korolar 2.6 daje uvjet pod kojim su te dvije veličine jednake. Središnji rezultat teorije mrežnog protoka koji potječe od Forda i Fulkersona je da su ove dvije veličine uvijek jednake. Ovaj rezultat, dokazan 1950-ih godina, doveo je do istraživanja na temu mrežnog protoka.

Teorem 2.8 *Vrijednost maksimalnog protoka jednaka je kapacitetu minimalnog $s-t$ reza.*

Ovaj teorem se ponekad naziva Teorem o maksimalnom protoku/minimalnom rezu. Uskoro ćemo dokazati ovaj teorem. Prije nego što to učinimo, potrebno je definirati pojam rezidualnog grafa. Neka je f protok na grafu $G = (V, A)$,

rezidualni graf u odnosu na protok f označen je s $G_f = (V, A)$. Luk $(i, j) \in A$ ima rezidualni kapacitet $u_f(i, j) = u(i, j) - f(i, j)$, tj. razliku između kapaciteta i protoka. Primijetimo da je rezidualni kapacitet uvijek nenegativan zbog ograničenja kapaciteta. Lukovi (i, j) s rezidualnim kapacitetom 0 nazivaju se zasićenim, tj. vrijednost protoka $f(i, j)$ jednaka je kapacitetu $u(i, j)$. Inače izostavljamo te lukove iz rezidualnog grafa u potpunosti, ali bit će korisno uključiti ih u naše dokaze. Označimo s A_f skup svih lukova s pozitivnim rezidualnim kapacitetom, tj. $A_f = \{(i, j) \in A : u_f(i, j) > 0\}$. Primijetimo da s našom definicijom protoka koja koristi obrnute lukove rezidualni kapacitet luka $u_f(j, i) = u(j, i) - f(j, i) = 0 + f(i, j) = f(i, j)$. Dakle, rezidualni kapacitet takvih obrnutih lukova (j, i) odgovara količini kojom možemo smanjiti pozitivni protok na luku (i, j). Kao primjer, na Slici 2.5 dajemo rezidualni graf povezan s protokom na Slici 2.3, izostavljamo lukove kojima je razidualni kapacit nula.



Slika 2.5: Rezidualni graf za protok sa Slike 2.3. Izostavljeni su bridovi kojima je rezidualni kapacitet bio nula. Postoji uvećavajući put $s - c - e - b - a - d - t$.

Ako postoji put P između vrhova s i t u rezidualnom grafu G_f na lukovima u A_f (tj. lukovima s pozitivnim rezidualnim kapacitetom) onda P nazivamo uvećavajućim putem. Pogledajmo Sliku 2.5 za primjer uvećavajućeg puta. Postojanje uvećavajućeg puta implicira da f nije maksimum jer možemo stvoriti novi protok $f^{'}$ veće vrijednosti uvećanjem protoka duž puta P. Formalno, neka je δ kapacitet rezidualnog luka s najmanjim rezidualnim kapacitetom na putu P, tj. $\delta = \min_{(i,j)\in P} u_f(i,j)$. Primijetimo da su svi lukovi na uvećavajućem putu s pozitivnim rezidualnim kapacitetom pa je $\delta > 0$. Tada stvaramo novi protok $f^{'}$ postavljanjem

$$f^{'}(i,j) = \begin{cases} f(i,j) + \delta & \text{ako } (i,j) \in P \\ f(i,j) - \delta & \text{ako } (j,i) \in P \\ f(i,j) & \text{inače, } (i,j),(j,i) \notin P \end{cases}$$

Ponekad kažemo da provodimo δ jedinica protoka duž puta P što rezultira protokom $f^{'}$. Također, ponekad nazivamo δ rezidualnim kapacitetom puta P. Potrebno je provjeriti je li $f^{'}$ i dalje protok prema Definiciji 2.4. Možemo pretpostaviti da je P jednostavan put. Očito su zadovoljeni uvjeti kapaciteta jer prema definiciji δ, za sve $(i,j) \in P$ vrijedi:

$$f^{'}(i,j) = f(i,j) + \delta \leq f(i,j) + (u_f(i,j) - f(i,j)) = u(i,j).$$

Simetrija lukova ostaje zadovoljena za sve lukove koji nisu na putu, dok za $(i,j) \in P$ imamo:

$$f^{'}(i,j) = f(i,j) + \delta = -(f(j,i) - \delta) = -f^{'}(j,i).$$

Uvjet očuvanja protoka ostaje zadovoljen za sve vrhove i koji nisu na putu. Ako se $i \neq s,t$ nalazi na putu onda postoje lukovi (h,i) i (i,j) na putu tako da je $f^{'}(i,j) = f(i,j) + \delta$ i $f^{'}(i,h) = f(i,h) - \delta$ pa imamo

$$\sum_{k:(i,k)\in A} f^{'}(i,k) = \sum_{k:(i,k)\in A} (f(i,k) + \delta - \delta) = 0.$$

Slično, ako je luk (s, j) prvi luk na putu onda je $f^{'}(s, j) = f(s, j) + \delta$ pa imamo

$$|f^{'}| = \sum_{k:(s,k)\in A} f^{'}(s, k) = \sum_{k:(s,k)\in A} (f(s, k) + \delta) = |f| + \delta.$$

Dakle, $f^{'}$ je protok veće vrijednosti od f. Možemo napokon dokazati Teorem 2.8 koji daje dvije ekvivalentne tvrdnje tvrdnji da je f maksimalni protok.

Teorem 2.9 *Sljedeće tvrdnje su ekvivalentne:*

1. *f je maksimalan mrežni protok G*
2. *rezidualna mreža G_f ne sadrži uvećavajući put*
3. *$|f| = u(\delta^+(S))$ za neki rez S*

Dokaz. Već smo pokazali da (1) $\implies$ (2) budući da smo prethodno pokazali da ako postoji uvećavajući put u G_f tada f nije maksimum.
Kako bismo pokazali da (2) $\implies$ (3) neka je S skup vrhova koji su dohvatljivi iz s s pozitivnim rezidualnim kapacitetima. Budući da nema uvećavajućih puteva onda $t \notin S$. Također, za svaki luk $(i, j) \in A$ takav da je $i \in S$ i $j \notin S$ mora biti $u_f(i, j) = 0$ i stoga je $f(i, j) = u(i, j)$. Tada prema Korolaru 2.6 slijedi $|f| = u(\delta^+(S))$.
Naposlijetku, kako bismo pokazali da (3) $\implies$ (1), sjetimo se da Lema 2.5 tvrdi da je $|f| \leq u(\delta^+(S))$ za svaki s-t rez S i svaki protok f tako da $|f| = u(\delta^+(S))$ implicira da f mora biti maksimalni protok, a S minimalni s-t rez. ■

Teorem 2.9 nas direktno dovodi do ideje kako implementirati dani algoritam što prikazujemo u Algoritmu 12. Postavljamo protok $f = 0$, a zatim tražimo uvećavajući put i onda ažuriramo protok. Ovaj nas algoritam vodi do

zaključka: ako su svi kapaciteti $u(i,j)$ cijeli brojevi tada postoji maksimalni protok takav da su i svi $f(i,j)$ cijeli brojevi i Algoritam 12 pronalazi takav protok f. Ako su svi $f(i,j)$ cijeli brojevi onda kažemo da je f *integral*. Budući da su inicijalno svi $f(i,j)$ cijeli brojevi, a onda ako su i svi $u(i,j)$ cijeli brojevi slijedi da su i svi rezidualni kapaciteti $u_f(i,j)$ cijeli brojevi pa je stoga δ cijeli broj, tako da novi protok f' ima sve $f'(i,j)$ cijele brojeve. Ovaj je zaključak jako koristan i često se naziva svojstvom integralnosti problema maksimalnog protoka.

Algorithm 12 Algoritam za uvećavajući put za maksimalni protok

$f(i,j) \leftarrow 0$ for all $(i,j) \in A$
while there is an augmenting path P in A_f **do**
    Push flow along P
    Update f
end while
return f

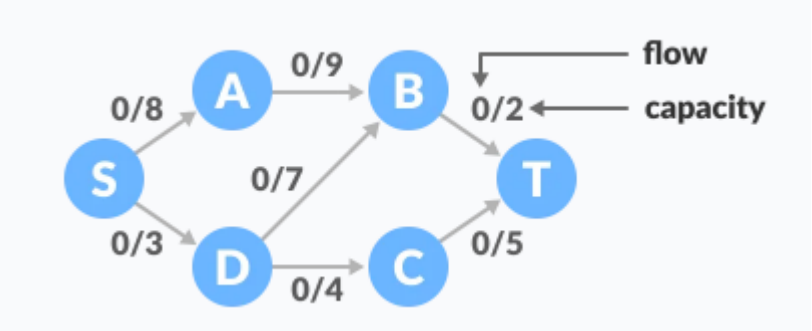
Lema 2.10 *Ako su svi kapaciteti $u(i,j)$ cijeli brojevi onda postoji integralni maksimalni protok f.*

Ako su kapaciteti cijeli brojevi onda možemo ograničiti broj iteracija algoritma (tj. broj puta kada pronalazimo uvećavajuće puteve) s $\mathcal{O}(mU)$ gdje je U maksimalni kapacitet, tj. $U = \max_{(i,j)\in A} u(i,j)$ (sjetite se da je m broj lukova, a n broj vrhova). Budući da je maksimalni protok protok izvorom, u najgorem slučaju u maksimalnom protoku imamo svih m lukova izvora s protokom jednakim maksimalnom kapacitetu tako da je vrijednost maksimalnog protoka najviše mU. Ako su svi kapaciteti cijeli brojevi onda u svakoj iteraciji uvećavamo vrijednost protoka za barem 1, početna vrijednost protoka je nula i može biti najviše mU stoga slijedi ograničenje broja iteracija.

Možemo pronaći put uvećanja u grafu u vremenu $\mathcal{O}(m)$, pa to dovodi do ukupnog vremena izvođenja od $\mathcal{O}(m^2U)$ (pretpostavljamo da je G povezan tako da je $m \geq n - 1$).

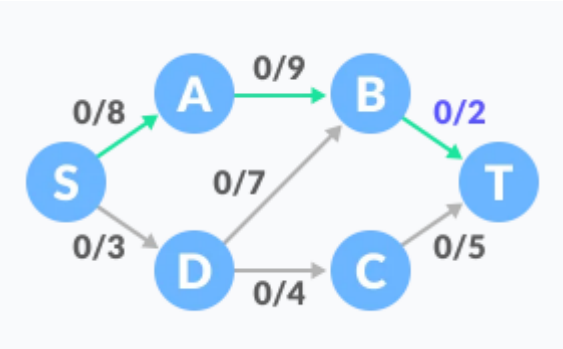
2.1.1 Primjer Ford-Fulkerson algoritma

Ford-Fulkerson algoritam jedan je oblik pohlepnog algoritma koji pronalazi maksimalan mrežni protok. Često ga nazivamo metodom budući da pristup pronalasku bridova u kojima uvećavamo protok u rezidualnom grafu nije u potpunosti određen. Algoritam su objavili Lester Randolph Ford jr. i Delbert Ray Fulkerson 1956. godine. To je jedan od najpoznatijih algoritama za rješavanje problema mrežnog protoka. Pokažimo na primjeru kako radi Ford-Fulkerson algoritam za mrežu sa Slike 2.6.



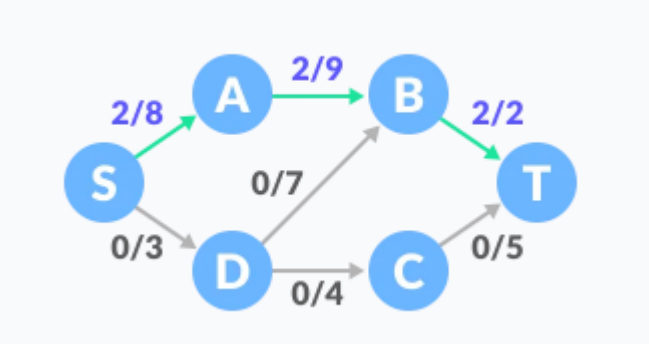
Slika 2.6: Graf mrežnog protoka.

1. korak: Odabiremo proizvoljan put od S do T. U ovom koraku smo odabrali put $S - A - B - T$ što je prikazano na Slici 2.7.



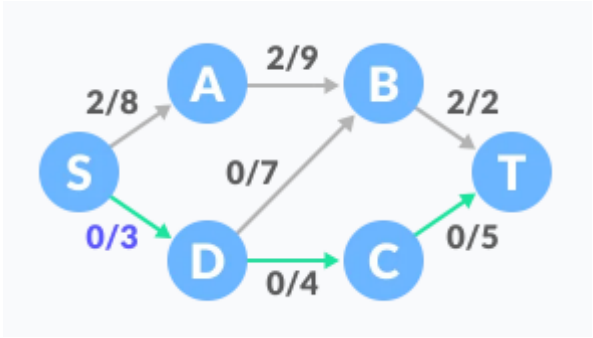
Slika 2.7: Pronalazak puta.

Najmanji kapacitet brida na izabranom putu je 2 $(B - T)$. Sukladno tome potrebno je ažurirati *protok*/*kapacitet* za svaki brid na putu što prikazujemo na Slici 2.8.



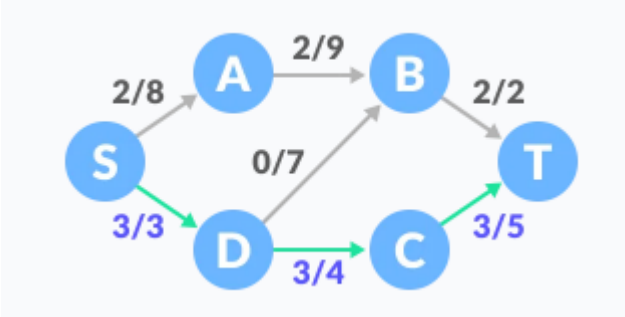
Slika 2.8: Graf nakon što smo ažurirali kapacitete.

2. korak: Sada odabiremo put $S - D - C - T$. Minimalni kapacitet brida na ovom putu je 3 što je prikazano na Slici 2.9.



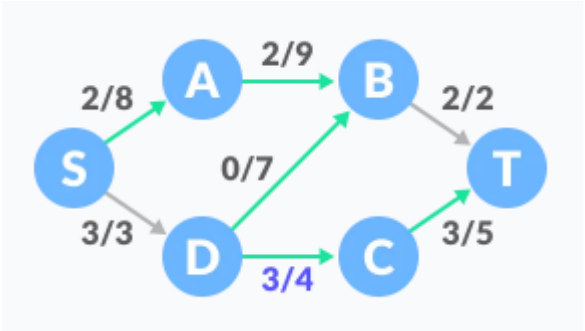
Slika 2.9: Pronalazak sljedećeg puta.

Zatim ponovno ažuriramo kapacitete kao što je na Slici 2.10.



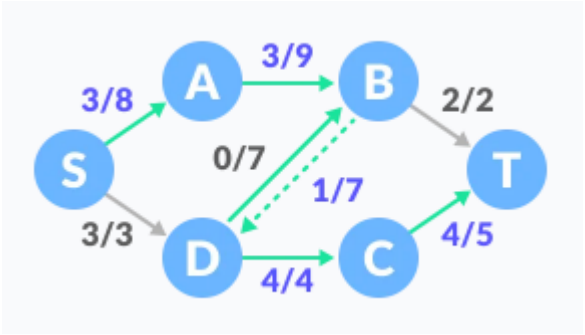
Slika 2.10: Graf nakon što smo ažurirali kapacitete.

3. Sada razmotrimo i obrnuti put $B-D$. Odabiremo putanju $S-A-B-D-C-T$ na Slici 2.11. Minimalni preostali kapacitet među bridovima je 1 $(D-C)$.



Slika 2.11: Pronalazak sljedećeg puta.

Zatim ponovno ažuriramo kapacitete. 2.12



Slika 2.12: Graf nakon što smo ažurirali kapacitete.

Na Slici 2.12 vidimo da više ne možemo poslati protok kroz niti jedan put. Maksimalan mrežni protok koji smo pronašli iznosi $|f| = 6$ jedinica protoka. Sada navedimo primjere primjena maksimalnog protoka.

2.2 Primjena maksimalnog mrežnog protoka

2.2.1 Dijeljenje prijevoza

Prva primjena uključuje pravednu raspodjelu vožnje skupa ljudi koji će zajedno putovati odabranim danima. Svaki tjedan ljudi u zajedničkom prije-

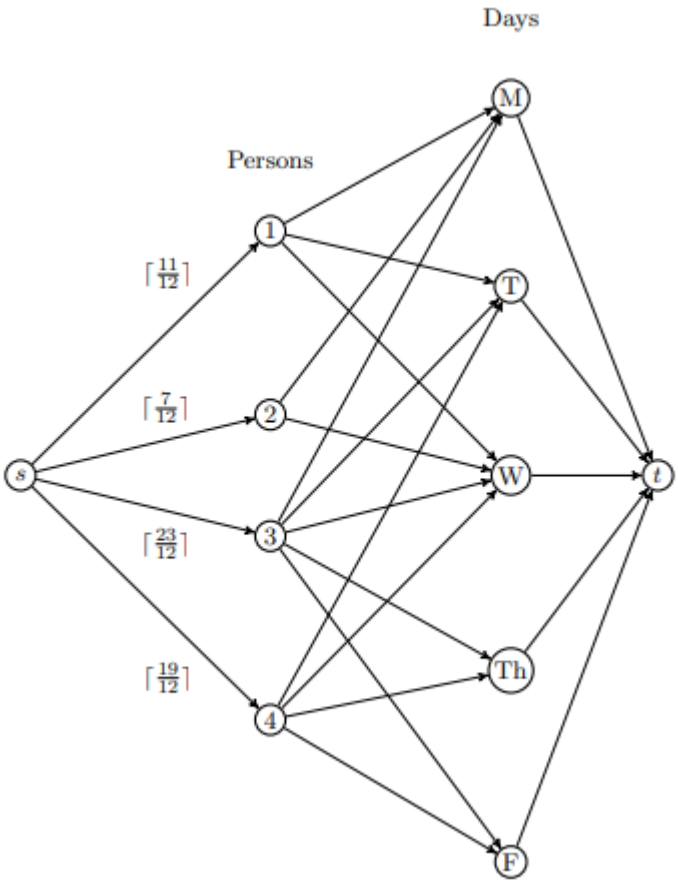
vozu objavljuju koje će dane koristiti zajednički prijevoz. Željeli bi smisliti način pravedne raspodjele vozačkih odgovornosti i naišli su na sljedeću ideju. U danu u kojem k ljudi koristi zajedničko vozilo, svaka će osoba dobiti $\frac{1}{k}$ udjela odgovornosti za vožnju. Neka je r_i ukupni udio i-te osobe za tjedan. Tada bi svaka osoba trebala voziti najviše $\lceil r_i \rceil$ puta taj tjedan. Na primjer, ovdje je ogledni tjedan s četiri osobe kao što je prikazano na Slici 2.13.

	M	T	W	Th	F
1	X	X	X		
2	X		X		
3	X	X	X	X	X
4		X	X	X	X

Slika 2.13: Ogledni tjedan s četiri osobe.

Osobe 1, 2 i 3 dobivaju $\frac{1}{3}$ za prijevoz u ponedjeljak, osobe 1, 3 i 4 dobivaju $\frac{1}{3}$ za prijevoz u utorak, osobe 1, 2, 3 i 4 dobivaju $\frac{1}{4}$ u srijedu i osobe 3 i 4 dobivaju po $\frac{1}{2}$ za čevrtak i petak. Konačno dobivamo: $r_1 = \frac{1}{3} + \frac{1}{3} + \frac{1}{4} = \frac{11}{12}$, $r_2 = \frac{1}{3} + \frac{1}{4} = \frac{7}{12}$, $r_3 = \frac{1}{3} + \frac{1}{3} + \frac{1}{4} + \frac{1}{2} + \frac{1}{2} = \frac{23}{12}$ i $r_4 = \frac{1}{3} + \frac{1}{4} + \frac{1}{2} + \frac{1}{2} = \frac{19}{12}$. Uočimo sad kako je $\sum_i r_i = 5$. $\sum_i r_i$ daje zbroj djelomičnih odgovornosti za svaki od pet dana. Sada postavljamo problem maksimalnog protoka kako bismo odlučili tko bi trebao voziti kojim danom. Da bismo to učinili postavili smo graf u kojem postoji jedan vrh po osobi i jedan vrh po danu u tjednu i njima dodajemo izvorni vrh s i poniručći vrh t. Dodajemo luk kapaciteta $\lceil r_i \rceil$ od izvora s do svake osobe i i luk kapaciteta 1 od svakog dana u tjednu do ponora t. Zatim za svaku osobu dodajemo luk kapaciteta 1 od osobe do dana

u tjednu kada osoba koristi zajednički prijevoz. Za dani primjer odgovarajuća instanca maksimalnog protoka pojavljuje se na Slici 2.14.



Slika 2.14: Instanca mreže protoka za problem zajedničkog putovanja. Svi neoznačeni lukovi imaju kapacitet 1.

Tvrdimo da ako je mrežni protok f vrijednosti 5 tako da je f *integral* onda postoji način za pravilnu raspodjelu odgovornosti za vožnju. Primijetimo da rez $S = V \setminus \{t\}$ ima kapacitet 5 (za pet lukova kapaciteta 1 od dnevnih vrhova do t) pa ako takav protok postoji to je maksimalni protok. Stoga svaki luk od dnevnog vrha do t mora biti zasićen (podsjetimo se da to znači da je protok jednak kapacitetu) pa je protok na luku 1. Za svaki dnevni

vrh znamo da je protok koji ulazi u vrh jednak protoku koji napušta vrh, a f je integral tako da mora postojati pozitivan protok vrijednosti 1 koji ulazi u svaki dnevni vrh koji dolazi od vrha neke osobe koja taj dan koristi zajednički prijevoz. Možemo dodijeliti odgovornost vožnje za dan odgovarajućoj osobi čiji luk prema dnevnom vrhu ima pozitivan protok. Budući da je protok koji napušta svaki vrh osobe jednak protoku koji ulazi u svaki vrh osobe, a protok koji ulazi u vrh osobe i je najviše $\lceil r_i \rceil$ (prema ograničenju kapaciteta na jednom luku koji ulazi u vrh osobe i) osobi i može se dodijeliti najviše $\lceil r_i \rceil$ dana što je ono što smo htjeli. Sada ostaje pokazati da je mrežni protok f vrijednosti 5 tako da je f integral. Umjesto toga dat ćemo protok vrijednosti 5, a zatim po svojstvu integralnosti zaključujemo: budući da postoji maksimalni protok vrijednosti 5 i svi kapaciteti su cijeli brojevi onda mora postojati integralni maksimalni protok te vrijednosti. Naš protok odgovara djelomičnom dodjeljivanju odgovornosti koje smo napravili na početku. Konkretno, za određeni dan, ako je k osoba koristilo zajednički prijevoz svakoj smo dodijelili udio od $\frac{1}{k}$ tako da smo u mreži za taj dan stavili protok vrijednosti $\frac{1}{k}$ na svaki luk od vrha osobe koristeći zajednički prijevoz tog dana do odgovarajućeg dnevnog vrha. Budući da je zbroj protoka koji ulazi u dnevni vrh 1, postavljamo protok na 1 za luk od dnevnog vrha do ponora t. Primijetimo da je protok koji napušta svaki vrh osobe i točno r_i tako da postavljamo protok na luku od s do osobe na vrh i na r_i. Sada imamo izvediv protok vrijednost $\sum_i r_i = 5$.

2.2.2 Problem eliminacije bejzbola

Sada se okrećemo drugoj primjeni problema maksimalnog protoka u kojoj nije očito da je uključen protok ili mreža. Primjena je poznata pod nazivom problem eliminacije bejzbola. Razmotrimo sljedeći primjer momčadi iz

istočne divizije Američke lige (izostavljajući Tampa Bay Rays, kako bismo pojednostavili kasniju raspravu).

Team	Wins	Games to play	Remaining schedule			
			NYY	BOS	TOR	BAL
New York Yankees (NYY)	93	8	–	1	6	1
Boston Red Sox (BOS)	89	4	1	–	0	3
Toronto Blue Jays (TOR)	88	7	6	0	–	1
Baltimore Orioles (BAL)	86	5	1	3	1	–

Slika 2.15: Problem eliminacije bejzbola.

Stupci "preostalog rasporeda" pokazuju broj utakmica koje dani tim mora odigrati protiv ostalih timova u diviziji. Pretpostavljamo da niti jedna momčad nema preostalih utakmica izvan divizije. Reći ćemo da momčad pobjeđuje u diviziji ako pobijedi u više utakmica od bilo koje druge momčadi u diviziji. Momčad je eliminirana ako ne može pobijediti u diviziji s obzirom na bilo koji ishod preostalih utakmica. Na primjer Baltimore je eliminiran jer može pobijediti najviše 91 utakmicu, 86 koje je već osvojio plus svojih 5 preostalih utakmica, ali New York je već pobijedio u 93 utakmice. Također, može se pokazati da je Boston eliminiran. Boston može pobijediti najviše 93 utakmice: 89 utakmica koje je već dobio plus 4 preostale utakmice. Međutim, ili će New York pobijediti u najmanje 94 utakmice pobjedom u barem jednoj od svojih preostalih utakmica, ili će izgubiti sve svoje preostale utak-

mice, uključujući svih 6 utakmica odigranih protiv Toronta. U posljednjem slučaju, Toronto će pobijediti u najmanje $88 + 6 \geq 94$ utakmica. U svakom slučaju, momčad koja nije Boston pobjeđuje u diviziji. Dakle, Boston je eliminiran. Pokazat ćemo da možemo odlučiti hoće li određeni tim biti eliminiran izračunavanjem maksimalnog protoka. Prvo moramo uvesti neke oznake za opći problem. Neka je T skup timova u diviziji. Neka je $w(i)$ broj pobjeda koje je tim $i \in T$ do sada ostvario. Neka je $g(i)$ broj utakmica koje tim i ima još odigrati i neka je $g(i, j)$ broj utakmica koje tim i i tim j imaju odigrati jedni protiv drugih. Neka je $w(R)$ ukupan broj pobjeda timova $R \subset T$ pa je $w(R) = \sum_{i \in R} w(i)$. Neka je $g(R)$ broj utakmica koje ekipe u R mogu odigrati jedna protiv druge tako da je $g(R) = \frac{1}{2} \sum_{i \in R, j \in R} g(i, j)$ (dijelimo s 2 jer zbroj dvostruko broji svaku igru). Konačno, za $R \subset T$ neka je $a(R) = \frac{1}{|R|}(w(R) + g(R))$.

Lema 2.11 *Za bilo koji $R \subset T$, neki tim $i \in R$ će pobijediti u najmanje $a(R)$ utakmica do kraja sezone.*

Dokaz. Ukupan broj pobjeda timova u R na kraju sezone je najmanje $w(R) + g(R)$. Neki tim u R pobjeđuje u svakoj od $g(R)$ utakmica koje se igraju između dva tima u R stoga prosječan broj pobjeda na kraju sezone za momčadi u R je najmanje $a(R)$, a neka momčad u R mora imati barem ovoliko pobjeda. ■

Korolar 2.12 *Ako za neki $k \in T$ i $R \subset T \setminus \{k\}$ vrijedi: $a(R) > w(k) + g(k)$ onda je tim k eliminiran.*

Dokaz. Budući da neki tim u R mora pobijediti više od $w(k) + g(k)$ igara i tim k može pobijediti najviše $w(k) + g(k)$ utakmica do kraja sezone, tim k ne može pobijediti u diviziji i stoga mora biti eliminiran. ■

Iz primjera poviše vrijedi: $R = \{NYY, TOR\}$, $k = BOS$, $w(R) = 93 + 88$,

$g(R) = 6$, $w(k) + g(k) = 89 + 4 = 93$, dok je $a(R) = \frac{1}{2}((93 + 88) + 6) = 93.5 > 93$. Zaključujemo da je Boston eliminiran.

Sada ćemo postaviti problem maksimalnog protoka kako bismo odlučili je li dani tim $k \in T$ eliminiran. Znamo da tim k nije eliminiran ako postoji neki ishod preostalih utakmica u kojima ima barem onoliko pobjeda koliko i svaka druga momčad. Neka su Z svi timovi osim k, tako da je $Z = T \setminus \{k\}$. Neka $x(i, j)$ predstavlja broj preostalih utakmica u kojima momčad i pobjeđuje momčad j. Jasno je da u svakoj od $g(i, j)$ preostalih utakmica između momčadi i i j vrijedi:

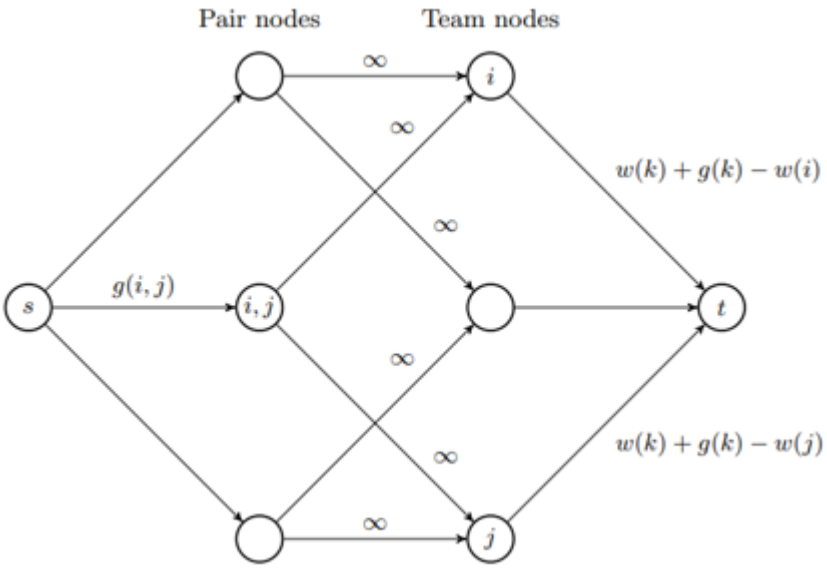
$$x(i, j) + x(j, i) = g(i, j).$$

Nadalje, možemo pretpostaviti da ako momčad k nije eliminirana, tada pobjeđuje u svim preostalim utakmicama i ima $w(k) + g(k)$ pobjeda. Bilo koji drugi tim i imat će $w(i) + \sum_{j \in Z} x(i, j)$ pobjeda (broj pobjeda koje trenutno ima plus broj preostalih utakmica koje pobjeđuje protiv timova u Z, podsjetimo, gubi sve utakmice od momčadi k). Dakle, da ekipa k ne bi bila eliminirana potrebno nam je:

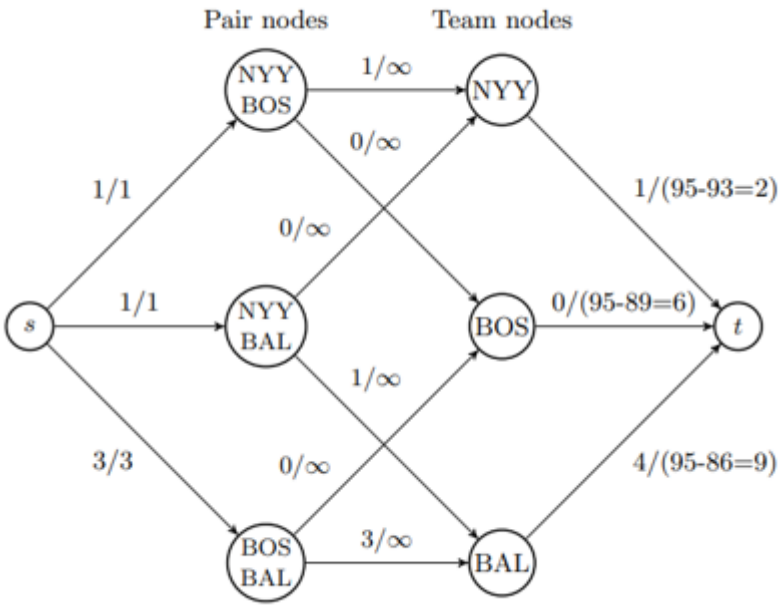
$$w(k) + g(k) \geq w(i) + \sum_{j \in Z} x(i, j), \forall i \in Z.$$

Konačno, trebamo da su $x(i, j)$ nenegativni cijeli brojevi. Stoga, ako možemo pronaći nenegativne cijele brojeve $x(i, j)$ tako da se poštuju prethodna dva uvjeta onda tim k nije eliminiran. Postavit ćemo problem maksimalnog protoka tako da ili pronađemo takav nenegativni cijeli broj $x(i, j)$ ili nalazimo skup $R \subset Z$ s $a(R) > w(k) + g(k)$. U prvom slučaju, postoji ishod igre takav da momčad k nije eliminirana, dok je u drugom slučaju tim k eliminiran po Korolaru 2.12. Postavljamo problem maksimalnog protoka. Imat ćemo početni vrh s, završni vrh t, jedan vrh za svaki par različitih timova u Z

(vrhovi parova) i jedan vrh za svaki tim u Z(vrhovi tima). Iz svakog vrha tima $i \in Z$ uključujemo luk do ponora kapaciteta $w(k)+g(k)-w(i)$. Ukoliko je $w(i) > w(k) + g(k)$ tim k je već eliminiran stoga pretpostavljamo da je $w(i) \leq w(k) + g(k)$ tako da je ovaj kapacitet nenegativan. Od izvornog vrha s do svakog para vrhova za par i, j uključujemo luk kapaciteta $g(i, j)$, tj. broj preostalih utakmica između timova i i j. Od parova vrhova za par i, j uključujemo jedan luk do vrha koji ima tim i beskonačnog kapaciteta, a drugi luk do vrha tima za j beskonačnog kapaciteta. Graf je prikazan na Slici 2.16. Na Slici 2.17 prikazujemo instancu protoka koja odgovara našem primjeru iznad provjeravajući je li Toronto eliminiran. Također dajemo maksimalni protok.



Slika 2.16: Ilustracija instance maksimalnog protoka za problem eliminacije u bejzbolu.



Slika 2.17: Ilustracija instance maksimalnog protoka za naš primjer problema eliminacije u bejzbolu. Provjera je li Toronto eliminiran.

Primijetimo da je $g(Z)$ broj utakmica odigranih između svih timova u Z, osim tima k. Sada možemo pokazati da tim k nije eliminiran ako i samo ako postoji protok vrijednosti $g(Z)$ u instanci protoka.

Lema 2.13 *Ako postoji protok vrijednosti $g(Z)$ u instanci maksimalnog protoka onda tim k nije eliminiran.*

Vraćajući se na naš primjer i protok na Slici 2.17, za $Z = NYY, BAL, BOS$, postoji protok vrijednosti $1 + 1 + 3 = 5$, i stoga Toronto nije eliminiran. Ovo slijedi ako New York pobijedi svoju jednu utakmicu protiv Bostona i izgubi svoju jednu utakmicu protiv Baltimorea, a Baltimore dobije sve 3 utakmice protiv Bostona. Zatim New York ima $93 + 1 = 94$ pobjede, Baltimore ima $86 + 4 = 90$ pobjeda, a Boston $89 + 0 = 89$ pobjeda. Ako Toronto dobije sve svoje preostale utakmice ima 95 pobjeda. Dakle, postoji neki ishod

preostalih utakmica u kojima Toronto ima barem onoliko pobjeda koliko i ostale momčadi u diviziji. Primijetimo da mogu postojati drugi protoci koji su također maksimalni i koji odgovaraju drugim mogućim scenarijima u kojima Toronto nije elimininiran. Sada moramo dokazati drugi smjer. To bismo mogli učiniti izravno tvrdeći da bilo koji ishod igara u kojima momčad k nije eliminirana dovodi do protoka vrijednosti $g(Z)$. Međutim, umjesto toga dajemo dokaz temeljen na vrijednosti minimalnog $s-t$ reza i Korolara 2.12.

Lema 2.14 *Ako je vrijednost maksimalnog protoka manja od $g(Z)$ onda se tim k eliminira.*

Dokaz. Kako je vrijednost maksimalnog protoka jednaka kapacitetu minimalnog $s-t$ reza i ako je vrijednost maksimalnog protoka manja od $g(Z)$ onda postoji minimalni $s-t$ rez S kapaciteta manjeg od $g(Z)$. Primjećujemo da ako je par vrhova za i, j u S onda su oba timska vrha i i j u S, ako jedan ili oba timska vrha nisu u S onda je luk od para vrhova i, j do timskog vrha (ili oba timska vrha) u presjeku definiranom sa S, a kapacitet presjeka je tada beskonačan što je kontradikcija. Neka je R skup svih timova čiji je timski vrh u S. Tada će lukovi u $\delta(S)^+$ uključivati one koji idu od timskih vrhova u R do ponora t, i one od izvora s do bilo kojeg para vrhova timova koji nisu oba u R jer ako je par vrhova i, j u S tada smo primijetili da i i i j moraju biti u R. Posljednji skup lukova ima ukupni kapacitet $g(Z) - g(R)$. Tako je kapacitet reza S najmanje:

$$g(Z)-g(R)+\sum_{i\in R}(w(k)+g(k)-w(i)) = g(Z)-g(R)+|R|(w(k)+g(k))-w(R).$$

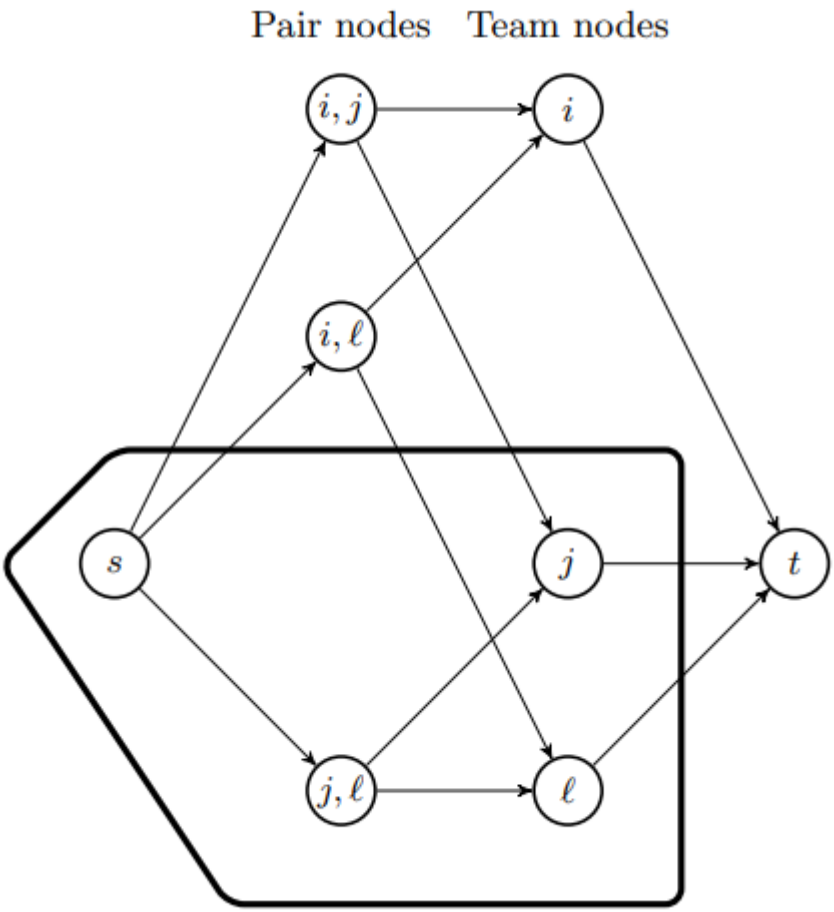
Pogledajmo Sliku 2.18. Po pretpostavci kapacitet reza je manji od $g(Z)$ pa je:

$$g(Z) - g(R) + |R|(w(k) + g(k)) - w(R) < g(Z),$$

ili

$$w(k) + g(k) < \frac{g(R) + w(R)}{|R|} = a(R).$$

Prema Korolaru 2.12 tim k je eliminiran. ■



Slika 2.18: Ilustracija $s-t$ reza u primjeru maksimalnog protoka za problem eliminacije bejzbola. Za ovaj rez, $R = \{j, l\}$.

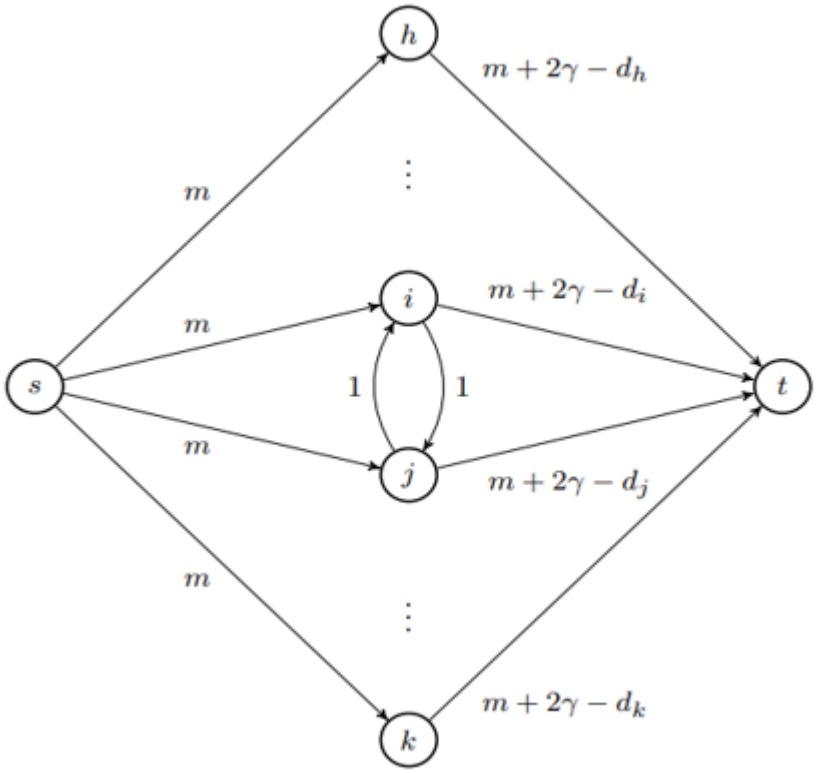
2.2.3 Pronalaženje podgrafa maksimalne gustoće

U ovom odjeljku obradit ćemo još jednu primjenu maksimalnog protoka gdje nije očito da je uključen protok. Ovdje je dan neusmjereni graf $G = (V, A)$, a mi želimo pronaći gust podgraf od G. Za podskup vrhova $S \subset V$, neka $G(S) = (V, A(S))$ označava podgraf induciran skupom vrhova S. Skup bridova $A(S) \subset A$ je skup svih bridova kojima su obje krajnje točke brida u S, odnosno $A(S) = \{(i, j) \in A : i, j \in S\}$. Gustoća podgrafa $G(S)$ je omjer veličine njegovog skupa bridova i njegovog skupa vrhova, odnosno gustoća je $\frac{|A(S)|}{|S|}$. Željeli bismo pronaći skup $S \subset V, S \neq \emptyset$ maksimalne gustoće. Neka D^* bude vrijednost maksimalne gustoće, tj.

$$D^* = max_{S \subset V, S \neq \emptyset} \frac{|A(S)|}{|S|}.$$

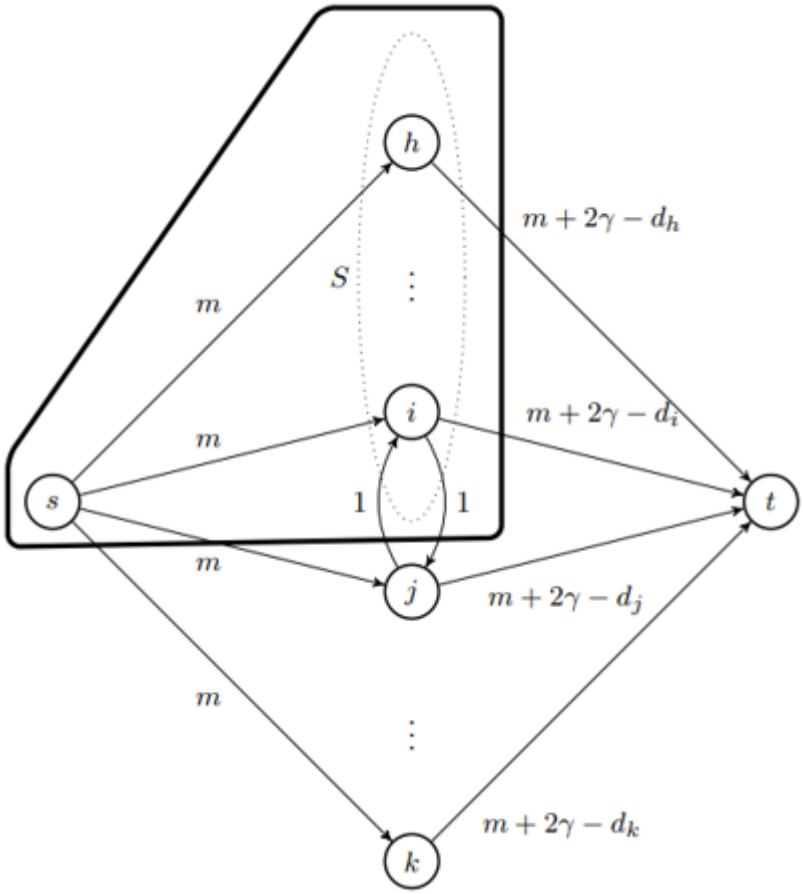
Neka je S^* skup vrhova za koji se postiže maksimum gustoće, tj. $D^* = \frac{|A(S^*)|}{|S^*|}$. Pretpostavimo da je G društvena mreža, a vrhovi predstavljaju ljude te neka postoji brid (i, j) ako su i i j prijatelji. Tada bi podgraf visoke gustoće odgovarao skupu ljudi koji su zajednički prijatelji možda zato što su svi dio iste škole ili neke druge slične organizacije. Automatsko pronalaženje takvih zajednica u društvenoj mreži bilo je područje intenzivnog istraživanja. U nastavku ćemo pokazati da možemo pronaći najgušći podgraf izvođenjem $\mathcal{O}(\log n)$ operacija maksimalnog protoka. Osnovna je ideja da ćemo započeti s pretpostavkom da je γ vrijednost D^*. Zatim ćemo koristiti izračun maksimalnog protoka da odredimo je li $\gamma \geq D^*$ ili ne. Zatim pomoću tehnike koja se zove bisekcijsko pretraživanje (ili binarno pretraživanje) možemo ažurirati vrijednost γ sve dok vrijednost γ ne bude dovoljno blizu D^* da možemo pronaći točnu vrijednost D^* i odgovarajući najgušći podgraf. S obzirom na pretpostavku o γ, konstruiramo instancu problema maksimalnog protoka u novom grafu G_0 kako slijedi. Kreiramo vrh i za svaki $i \in V$ i dodamo izvor

s i ponor t tako da je $V_0 = V \cup \{s, t\}$. Za svaki brid $(i, j) \in A$ dodajemo dva luka (i, j) i (j, i) kapaciteta 1. Za svaki $i \in V$ dodajemo luk (s, i) kapaciteta m i luk (i, t) kapaciteta $m+2\gamma-d_i$ gdje je d_i stupanj vrha i u izvornom grafu (tj. broj bridova incidentnih s i). Kako je $d_i \leq m$ kapacitet je nenegativan. Pogledajmo Sliku 2.19 za ilustraciju.



Slika 2.19: Ilustracija instance maksimalnog protoka za problem pronalaska najgušćeg podgrafa za pretpostavljeni γ.

Zatim izračunavamo maksimalni protok grafa G'. Kako bismo ograničili vrijednost maksimalnog protoka, pretpostavljamo da je $s - t$ rez $\{s\} \cup S$ za $S \subset V$. Kapacitet ovog reza uključuje sve lukove iz s do $i \in V \setminus S$, iz $i \in S$ do t, i iz $i \in S$ do $j \in V \setminus S$. Pogledajmo Sliku 2.20.



Slika 2.20: Ilustracija $s-t$ reza za $\{s\} \cup S$ za instancu maksimalnog protoka za problem pronalaska maksimalne gustoće podgrafa. Iscrtani dio daje vrhove u S iz početnog grafa.

$\delta(S)$ je oznaka za skup bridova u G tako da točno jedna krajnja točka brida bude u S. Promatramo $\sum_{i \in S} d_i = 2|A(S)| + |\delta(S)|$. Budući da $\sum_{i \in S} d_i$ dvostruko broji svaki brid koji ima obje krajnje točke u S i broji svaki brid

u $\delta(S)$ točno jednom. Onda je kapacitet $s-t$ reza $\{s\} \cup S$ jednak:

$$m|V-S| + |\delta(S)| + \sum_{i \in S}(m + 2\gamma - d_i) = \tag{2.1}$$

$$= mn - m|S| + |\delta(S)| + m|S| + 2\gamma|S| - \sum_{i \in S} d_i = \tag{2.2}$$

$$= mn + |\delta(S)| + 2\gamma|S| - (2|A(S)| + |\delta(S)|) = \tag{2.3}$$

$$= mn + 2|S|(\gamma - \frac{|A(S)|}{|S|}). \tag{2.4}$$

Sada možemo dokazati sljedeću Lemu 2.15.

Lema 2.15 *Vrijednost maksimalnog protoka je mn ako i samo ako je* $\gamma \geq D^*$.

Dokaz. Pretpostavimo da $s-t$ rez $\{s\}$ ima kapacitet mn pa iz toga slijedi da vrijednost maksimalnog protoka može biti najviše mn. Ako je $\gamma < D^*$ onda skup S^* ima kapacitet od odgovarajućeg $s-t$ reza $\{s\} \cup S^*$ koji je $mn + 2|S^*|(\gamma - \frac{|A(S^*)|}{|S^*|}) < mn$ prema jednadžbi (2.1) budući da je $S^* \neq \emptyset$. Prema tome maksimalni protok u ovom slučaju mora biti strogo manji od mn. Slično pretpostavimo da je vrijednost maksimalnog protoka manja od mn. Neka je $\{s\} \cup S$ minimalni $s-t$ rez, prema Teoremu 2.9 on ima jednak kapacitet maksimalnom protoku koji je manji od mn i stoga vrijedi $S \neq \emptyset$. Prema jednadžbi (2.1) kapacitet reza je $mn + 2|S|(\gamma - \frac{|A(S)|}{|S|}) < mn$ i onda mora vrijediti da je $\gamma < \frac{|A(S)|}{|S|} \leq D^*$. Time je proveden dokaz. ■

U nastavku ćemo pretpostaviti da je D' druga najveća gustoća tj. $D' < D^*$ pa je za svaki $S \neq \emptyset$ ili $\frac{|A(S)|}{|S|} = D^*$ ili $\frac{|A(S)|}{|S|} = D'$.

Korolar 2.16 *Ako je* $D' < \gamma < D^*$ *onda za minimalni* $s-t$ *rez* $\{s\} \cup X$ *koji odgovara maksimalnom protoku mora vrijediti da je* X *podgraf maksimalne gustoće.*

Dokaz. Prema jednadžbi (2.1) kapacitet bilo kojeg $s-t$ reza $\{s\} \cup S$ koji ima gustoću $D^{'} \leq \gamma$ je $mn + 2|S|(\gamma - \frac{|A(S)|}{|S|}) = mn + 2|S|(\gamma - D^{'}) \geq mn$ i to nije minimalni $s-t$ rez. Međutim, prema Lemi 2.15 vrijednost maksimalnog protoka je strogo manja od mn pa minimalni $s-t$ rez $\{s\} \cup X$ mora imati kapacitet $mn + 2|S|(\gamma - \frac{|A(X)|}{|X|}) < mn$ i onda je $\gamma < \frac{|A(X)|}{|X|}$. Budući da je γ barem druga najveća gustoća, mora biti da je X podgraf maksimalne gustoće. ■

Sada objašnjavamo kako iskoristiti izračun za maksimalni protok kao podproces za pronalazak γ koji se nalazi između $D^{'}$ i D^{*} tako da budemo sigurni da smo pronašli podgraf maksimalne gustoće D^{*}. Da bismo to učinili koristimo bisekcijsko pretraživanje (ili binarno pretraživanje). Tijekom pretraživanja održavamo interval $\langle l, u]$ tako da smo sigurni da je $D^{*} \in \langle l, u]$. Pretpostavimo da je $A \neq \emptyset$ (budući da $A = \emptyset$ onda je trivijalno $D^{*} = 0$). Možemo inicijalizirati interval na $\langle 0, m]$ budući da je jasno da je $0 < D^{*} \leq m$. Zatim izračunavamo maksimalni protok kako je navedeno u gornjoj instanci s pretpostavkom da je γ postavljena na sredinu intervala pa je $\gamma = \frac{u+l}{2}$. Prema Lemi 2.15 ako je vrijednost maksimalnog protoka mn onda je $D^{*} \leq \gamma = \frac{u+l}{2}$ pa onda znamo da je $D^{*} \in \langle l, \frac{u+l}{2}]$ i onda stavljamo vrijednost od u na $\frac{u+l}{2}$. Inače, ako je vrijednost maksimalnog protoka manja od mn onda je $D^{*} > \gamma = \frac{u+l}{2}$ pa je $D^{*} \in \langle \frac{u+l}{2}, u]$ i možemo staviti vrijednost od l na $\frac{u+l}{2}$. U oba slučaja vrijednost intervala se upola smanjila i još uvijek sadrži D^{*}. U nastavku ćemo pokazati da ako je interval $\langle l, u]$ dovoljno malen, tada možemo primijeniti Korolar 2.16 i osigurati da smo pronašli podgraf maksimalne gustoće.

Lema 2.17 *Ako je* $u - l < \frac{1}{n^2}$ *onda je* $\gamma = l$ *i* $D^{'} \leq \gamma < D^{*}$.

Algorithm 13 Algoritam za izračunavanje podgrafa maksimalne gustoće

if $A = \emptyset$ **then**

return i

Return an arbitrary $i \in V$

end if

$l \leftarrow 0$

$u \leftarrow m$

$X \leftarrow \emptyset$

while $(u - l) \geq \frac{1}{n^2}$ **do**

Compute maximum flow f, minimum $s - t$ cut $\{s\} \cup S$ in maximum flow instance with guess $\gamma = \frac{u+l}{2}$

if $|f| = mn$ **then**

$u \leftarrow \frac{u+l}{2}$

else if **then**

$l \leftarrow \frac{u+l}{2}$

$X \leftarrow S$

end if

return X

end while

Teorem 2.18 *Algoritam 13 pronalazi podgraf maksimalne gustoće u* $\mathcal{O}(\log n)$ *izračuna maksimalnog protoka.*

Dokaz. Ako je $A = \emptyset$ onda je svaki pojedinačni vrh najgušći podgraf i Algoritam 13 vraća jedan vrh. Inače, algoritam pretpostavlja da je $D^* \in \langle l, u]$ i da X odgovara minimalnom $s-t$ rezu $\{s\} \cup X$ dobivenom s pretpostavkom da je $\gamma = l$. Prema Lemi 2.17 i Korolaru 2.16 vrijedi $u - l < \frac{1}{n^2}$ i iz toga slijedi da je X podgraf maksimalne gustoće. Da bismo dovršili dokaz moramo odrediti broj ponavljanja *while* petlje. Početna veličina intervala $\langle l, u]$ je m

i algoritam se izvršava jednom u $u - l < \frac{1}{n^2}$ i u svakoj iteraciji veličina intervala se upola smanjuje. Prema tome potrebno je najvise $\lceil \log_2(\frac{m}{\frac{1}{n^2}}\rceil) = \mathcal{O}(\log mn^2) = \mathcal{O}(\log n^4) = \mathcal{O}(\log n)$ ponavljanja dok se algoritam ne izvrši jer ima najviše $\binom{n}{2} = \mathcal{O}(n^2)$ bridova u svakom neusmjerenom grafu. ■

2.3 Najbolji uvećavajući put

Počinjemo s vrlo prirodnom idejom, tražimo put uvećanja koji uvećava vrijednost protoka što je više moguće. Drugim riječima, želimo put uvećanja čiji je minimalni rezidualni kapacitet luka što je veći moguć, nazvat ćemo to najbolji uvećavajući put. Dajemo verziju ovog algoritma za pronalaženje puta uvećanja u Algoritmu 14.

Algorithm 14 Algoritam najboljeg uvećavajućeg puta za problem maksimalnog protoka

$f(i,j) \leftarrow 0, \forall\ (i,j) \in A$
while there is an augmenting path in A_f **do**
  Let P be augmenting path that maximizes $\min_{(i,j)\in P} u_f(i,j)$
  Push flow along P
  Update f
end while
return f

Definicija 2.19 *Kaže se da je algoritam pseudopolinomijalan ako broj operacija algoritma ima polinomnu gornju granicu u veličini ulaznih podataka ako su oni zapisani unarno.*

Osnovna ideja analize algoritma je prilično jednostavna. Pokazat ćemo da se razlika između maksimalnog protoka f^* i trenutnog protoka f može ras-

taviti na najviše m uvećavajućih puteva. Budući da uvijek uvećavamo duž najboljeg takvog puta, put uvećava $|f|$ barem za faktor $\frac{1}{m}$ od $(|f^*| - |f|)$, tj. za razliku između maksimalnog i trenutnog protoka. Možemo pokazati da se nakon m takvih uvećanja razlika smanjila za konstantni faktor što nam omogućuje da pokažemo da nakon polinomno ograničenog broja proširenja trenutni protok mora biti maksimalan. Ova osnovna ideja prikazivanja dekompozicije na m objekata i pokazivanja da algoritam izvodi ažuriranje barem jednako dobro kao bilo koji od ovih objekata, dovodi do uvećanja za konstantni faktor nakon m ažuriranja. Sada ćemo razraditi detalje ove ideje. Prvo dokazujemo Lemu o dekompoziciji protoka 2.21 koja pokazuje da se svaki protok može rastaviti na protoke sa najviše m $s-t$ putanja i ciklusa. U nastavku za protoke f, f' i f'' pišemo $f = f' + f''$ (odnosno $f = f' - f''$) ako $f(i,j) = f'(i,j) + f''(i,j)$ (odnosno, $f(i,j) = f'(i,j) - f''(i,j)$) za sve lukove $(i,j) \in A$. Sljedeća Lema 2.20 je jednostavna, ali korisna.

Lema 2.20 *Ako protoci f' i f'' zadovoljavaju svojstva očuvanja protoka i simetrije iskrivljenja onda ista svojstva vrijede i za protok $f = f' + f''$ i $f = f' - f''$. Također, apsolutna vrijednost protoka se ponaša na sljedeći način: $|f| = |f' + f''| = |f'| + |f''|$ i $|f| = |f' - f''| = |f'| - |f''|$.*

Sada ćemo iznijeti Lemu o dekompoziciji protoka 2.21.

Lema 2.21 *Za zadani s-t protok f, postoje protoci $f_1, ..., f_l$, za neki $l \leq m$, takvi da vrijedi $f = \sum_{i=1}^{l} f_i$, $|f| = \sum_{i=1}^{l} |f_i|$, i za svaki i, lukovi protoka f_i s pozitivnim protokom oblikuju ili jednostavan s-t put ili ciklus.*

Dokaz. Dokazujemo tvrdnju indukcijom na broj lukova s pozitivnim protokom. Zapravo dokazujemo jaču tvrdnju za l, broj lukova s pozitivnim protokom. Očito je $l \leq m$.

Ako je $l = 0$ onda je tvrdnja trivijalno istinita. Pretpostavimo da tvrdnja

vrijedi za $l < p$ i da protok f ima $l = p$ lukova s pozitivnim protokom. Odaberimo bilo koji luk $(i, j) \in A$ takav da je $f(i, j) > 0$. Ako je $i \neq s$ onda prema principu očuvanja protoka postoji neki vrh h takav da je $f(h, i) > 0$. Slično tome, ako je $j \neq t$ onda postoji neki vrh k takav da je $f(j, k) > 0$. Možemo nastaviti ovaj argument dok ne dobijemo ili jednostavni $s - t$ put P lukova koji imaju pozitivan protok ili ciklus C lukova koji imaju pozitivan protok. Pretpostavimo da imamo jednostavni $s-t$ put P, slučaj za ciklus C je sličan. Postavljamo $\delta = \min_{(i,j)\in P} f(i, j)$. Također, postavljamo $f_p(i, j) = \delta$, $f_p(j, i) = -\delta$ za $(i, j) \in P$, i $f_p(i, j) = 0$ inače. Tvrdimo da je lako provjeriti da je f_p protok. Neka je $f' = f - f_p$. Tada f' ima barem jedan luk manje s pozitivnim protokom nego f (posebno, luk $(i, j) \in P$ za koji vrijedi $f(i, j) = \delta$). Prema Lemi 2.20, f' zadovoljava očuvanje protoka i svojstvo simetrije iskrivljenja jer to čine f i f_p, f' također zadovoljava ograničenja kapaciteta jer za $f(i, j) > 0$, $f'(i, j) = f(i, j) - f_p(i, j) \leq f(i, j) \leq u(i, j)$, i prema svojstvu simetrije iskrivljenja za $f(i, j) < 0$, $f'(i, j) = f(i, j) - f_p(i, j) \leq 0 \leq u(i, j)$. Dakle, f' je protok s najviše $p - 1$ lukova pozitivnog protoka i može se zapisati kao $f' = \sum_{i=1}^{p-1} f_i$ indukcijom. Stoga je $f = f' + f_p = \sum_{i=1}^{p} f_i$ i tvrdnja leme slijedi. ■

Sljedeće, trebamo pokazati da možemo primijenimo ovu dekompozicijsku Lemu 2.21 ne na protoke u originalnom grafu G, već na protoke u rezidualnom grafu G_f s rezidualnim kapacitetima u_f te da protoci u rezidualnom grafu imaju prirodan odnos prema protocima u G.

Lema 2.22 *Neka je f $s - t$ protok u grafu G i neka je f^* maksimalni $s - t$ protok u grafu G. Tada maksimalni protok u rezidualnom grafu G_f ima vrijednost $|f^*| - |f|$.*

Sada ćemo pokazati da je rezidualni kapacitet najboljeg uvećavajućeg puta relativno velik kombinirajući prethodne dvije Leme 2.22 2.21.

Lema 2.23 *Neka je f^* maksimalni protok u G, a f bilo koji $s-t$ protok. Tada je rezidualni kapacitet najboljeg uvećavajućeg puta barem $\frac{1}{m}(|f^*| - |f|)$.*

Sada možemo koristiti Lemu 2.23 kako bismo ograničili broj iteracija algoritma najboljeg uvećavajućeg puta. Kako bismo dali ograničenje broja iteracija, neka U predstavlja najveći kapacitet svih lukova, tj. $U = \max_{(i,j)\in A} u(i,j)$.

Teorem 2.24 *Ako su kapaciteti cjelobrojni, Algoritam 14 računa maksimalni protok u $\mathcal{O}(m \ln(mU))$ iteracija.*

Dokaz. Očito ograničenje vrijednosti maksimalnog protoka je mU, potencijalno svaki luk izlazi iz izvora i zasićen je maksimalnim kapacitetom U. Ako je f protok u nekoj iteraciji algoritma, neka je f^k rezultirajući protok nakon k iteracija. Tada prema Lemi 2.23 slijedi:

$$|f^{(1)}| \geq |f| + \frac{1}{m}(|f^*| - |f|)$$

ili

$$|f^*| - |f^{(1)}| \leq \left(1 - \frac{1}{m}\right)(|f^*| - |f|).$$

Slično vrijedi:

$$|f^{(2)}| \geq |f^{(1)}| + \frac{1}{m}(|f| - |f^{(1)}|)$$

ili

$$|f^*| - |f^{(2)}| \leq \left(1 - \frac{1}{m}\right)\left(|f^*| - |f^{(1)}|\right) \leq \left(1 - \frac{1}{m}\right)^2 (|f^*| - |f|).$$

Općenito vrijedi:

$$|f^*| - |f^{(k)}| \leq \left(1 - \frac{1}{m}\right)^k (|f^*| - |f|)$$

Tako da nakon $k = m\ln(mU)$ iteracija, počevši s protokom $f = 0$, imamo sljedeće:

$$|f^*| - |f^{(k)}| \leq \left(1 - \frac{1}{m}\right)^{m\ln(mU)} (|f^*| - |f|) < e^{-\ln(mU)}|f^*|$$

Koristeći $1 - x < e^{-x}$ za $x \neq 0$. Slijedi:

$$|f^*| - |f^{(k)}| \leq e^{-\ln(mU)}|f^*| = \frac{1}{mU}|f^*| \leq 1.$$

Prema svojstvu cjelobrojnosti, budući da su svi kapaciteti cijeli brojevi, vrijednost $|f^*|$ je također cijeli broj. Nadalje, prema svojstvima algoritma uvećavajućeg puta, $|f^{(k)}|$ je cijeli broj. Dakle, ako je $|f^*| - |f^{(k)}| < 1$ onda je $|f^{(k)}| = |f^*|$, i $f^{(k)}$ je maksimalni protok. ■

2.4 Najkraći uvećavajući put

U ovoj varijanti, uvijek proširujemo duž najkraćeg uvećavajućeg puta, odnosno puta s najmanjim brojem lukova. Algoritam za pronalazak najkraćeg uvećavajućeg puta je prikazan u Algoritmu 15.

Algorithm 15

$f(i,j) \leftarrow for\ all\ (i,j) \in A$

while there is an augmenting path in A_f **do**

  Let P be a shortest augmenting path in A_f

  Push flow along P

  Update f

end while

return f

Neka je $d(i)$ broj lukova u najkraćoj putanji od vrha $i \in V$ do ponora t u trenutnom rezidualnom grafu G_f koristeći samo lukove iz skupa A_f. Primijetimo da je ova ideja udaljenosti drugačija od ideje koju smo koristili u Poglavlju 1 gdje je $d(i)$ predstavljalo udaljenost od izvora s, dok ovdje predstavlja udaljenost do ponora t. Stoga je i naša ideja o ispravnim oznakama

udaljenosti također drugačija: ovdje znamo da za bilo koji luk (i, j) s pozitivnim preostalim kapacitetom mora vrijediti $d(i) \leq d(j) + 1$. Inače, ako je $d(i) > d(j) + 1$, tada postoji kraća putanja od i do t uzimanjem luka (i, j) i putanje od j do ponora. Posebno, imamo $d(i) = \min_{(i,j)\in A_f}(d(j)+1)$, i stoga za luk (i, j) koji se nalazi na najkraćoj putanji, vrijedi $d(i) = d(j) + 1$.

Lema 2.25 *Određeni luk $(i, j) \in A$ zasićuje se $\mathcal{O}(n)$ puta tijekom izvršavanja Algoritma 15.*

Teorem 2.26 *Algoritam 15 računa maksimalni protok u $\mathcal{O}(mn)$ iteracija.*

Dokaz. Prema Lemi 2.25, određeni luk može biti zasićen $\mathcal{O}(n)$ puta tijekom algoritma. Svaka iteracija algoritma za uvećanje protoka zasićuje barem jedan luk. Budući da postoji m lukova, može biti najviše $\mathcal{O}(mn)$ iteracija. ■

Literatura

[1] M. Axenovich, Lecture Notes Graph Theory, 2016.

[2] J.A. Bondy, U.S.R. Murty, Graph Theory With Applications, 1976.

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

[4] L.R. Ford Jr. i D.R. Fulkerson, Flows in Networks, Princeton University, 1962.

[5] David M. Mount, Design and Analysis of Computer Algorithms, Department of Computer Science University of Maryland, 2015.

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

[7] David P. Williamson, Network Flow Algorithms, Cornell University, 2019.

[8] `https://www.programiz.com/dsa/ford-fulkerson-algorithm`