SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Algoritmi i metode optimizacija

dokument u izradi, koristiti isključivo kao dodatak ostaloj literaturi

verzija 0.3

Josip Gracin

Zagreb, 2004.

Sadržaj

Uvod

Ova skripta namijenjena su studentima treće godine smjera Telekomunikacije i informatika, Fakulteta elektrotehnike i računarstva u Zagrebu kao pomoć pri svladavanju gradiva iz kolegija Algoritmi i metode optimizacija. Iako ovaj materijal predstavlja grubu, fragmentalnu i metodički neuređenu verziju konačnog dokumenta, uvršten je u popis materijala za ovaj kolegij s namjerom da se primjedbe čitatelja što prije uvrste u sâm materijal. Autor će, stoga, biti zahvalan na svim ispravkama, primjedbama i sugestijama koje se, između ostalog, mogu uputiti i na email `gracin@tel.fer.hr`.

Poglavlje 1

Složenost algoritama

Ukoliko za neki problem postoji više algoritama koji ga rješavaju, postavlja se pitanje koji je od algoritama najbolje iskoristiti. Da bismo mogli uspoređivati algoritme i odrediti najboljeg, potrebno je definirati mjeru kojom ocjenjujemo koliko je algoritam dobar.

Uzmimo, na primjer, niz algoritama za pronalaženje zadanog broja u sortiranom nizu brojeva. Mjera kojom bismo mogli uspoređivati algoritme mogla bi se definirati kao *vrijeme izvođenja algoritma* na nekom konkretnom problemu (dakle, problemu sa zadanim nizom brojeva i brojem kojeg je potrebno pronaći) i na nekom konkretnom računalu. Međutim, ovako definirana mjera imala bi niz nedostataka. Kao prvo, svaki algoritam koji bi htjeli usporediti s nekim od postojećih za koji je već pronađena mjera, morao bi se izvesti na istom problemu i na istom računalu. Ovo predstavlja veliko ograničenje jer bi usporedba algoritama bila ograničena samo na one algoritme čije vrijeme izvođenja je izmjereno na jednom konkretnom računalu. Drugi nedostatak je što na osnovi niza mjerenja za probleme određene veličine i dalje ne bismo mogli biti sigurni da je određeni algoritam koji je brži na izmjerenim problemima, ujedno brži i za proizvoljno velike probleme. Na primjer, algoritam koji brže od nekog drugog sortira niz od 500 brojeva, ne mora nužno biti brži i za niz od milijun ili više brojeva.

Mjera koja bi bila od koristi u praksi morala bi uzimati u obzir i ovisnost vremena izvršavanja algoritma o veličini ulaznog problema.

Nedovršeno **Nedovršeno**

Poglavlje 2

Strukture podataka

Organizacija podataka koji opisuju neki konkretni problem bitno utječe na svojstva algoritama koji rješavaju taj problem. Uz ispravnu organizaciju, podaci su uvijek "pri ruci" te se algoritam može jednostavnije definirati i često brže izvoditi. Neispravan izbor struktura podataka opterećuje algoritam nepotrebnim koracima prilagođavanja podataka samom algoritmu. Stoga je takve korake potrebno izbjeći na samom početku pri definiranju algoritma, izabirući prikladne strukture podataka.

U ovom poglavlju, ukratko su prikazane neke češće korištene strukture podataka i njihove realizacije.

2.1 Slijedna lista ili niz

Niz istovrsnih podataka moguće je u memoriji računala pohraniti sekvencijalno, tj. na način da se svaki sljedeći podatak u nizu smješta na memorijsku lokaciju neposredno iza prethodnog. Uz pretpostavku da su svi podaci jednake veličine (iznosa s) te da se u memoriju smještaju počevši od lokacije A, podatak koji je i-ti u slijedu smješta se na adresu

$$M_i = A + i \cdot s.$$

Bitno svojstvo ovakve strukture podataka je da se na temelju rednog broja podatka u slijedu, može vrlo brzo odrediti memorijska lokacija na koju je smješten.

Primjer 2.1 U nekom programskom sustavu često se koristi pomoćna funkcija za dobivanje n-tog po redu prostog broja. U velikoj većini slučajeva, traži se jedan od prvih 100 prostih brojeva. Potrebno je realizirati funkciju na način da se prvih sto prostih brojeva izračuna prilikom inicijalizacije sustava te se za vrijeme rada, ukoliko se traži neki od unaprijed izračunatih brojeva, on isporučuje bez proračuna.

Rješenje: (Naputak.) Izračunati prvih stotinu prostih brojeva i smjestiti ih u slijednu listu. Pri pozivu funkcije, provjeriti da li je traženi redni broj manji od 100 i u tom slučaju vratiti pripadajući broj iz liste. U suprotnom, izračunati broj.

Slijedne liste nisu pogodne za korištenje ukoliko je pojedine elemente potrebno iz liste brisati i umetati. Svako umetanje novog elementa između dva postojeća zahtijeva pomicanje svih elemenata s rednim brojem većim od onog koji se umeće, na veću memorijsku lokaciju. Isto tako, ukoliko se neki element (osim posljednjeg) iz liste briše, potrebno je sve elemente s većim rednim brojem pomaknuti na niže memorijske lokacije.

Primjer 2.2 transpose i move-to-front sekvencijalno pretraživanje

Nedovršeno **Nedovršeno**

2.2 Vezana lista

NEDOVRŠENO NEDOVRŠENO

2.3 Mapa

NEDOVRŠENO NEDOVRŠENO

2.4 Stablo

NEDOVRŠENO NEDOVRŠENO

2.5 Raspršeno adresiranje

Zadatak. U dijelu kôda koji upravlja Ethernet mrežnom karticom u nekom operacijskom sustavu potrebno je, s obzirom na oznaku protokola kojem primljeni okvir pripada, pozivati odgovarajuću funkciju zaduženu za pojedini protokol. Protokoli koje je potrebno podržati i njihove oznake dani su u tablici (svi brojevi su heksadecimalni). S obzirom da na mrežnu karticu dolaze tisuće paketa u sekundi, kôd koji izabire funkciju za obradu mora biti što je moguće efikasniji.

Protokolni brojevi (*protocol numbers*)

IP	0800
IEEE 802.3	0001
AX.25	0002
IEEE 802.2	0004
RARP	8035
ARP	0806
IPX	8137
Localtalk	0009
IPv6	86DD
BPQ	08FF

Rješenje:

Problem bi bilo moguće riješiti na sljedeći način.

```
struct packet *input_packet;
...
switch ( protocol )
{
case 0x0800:
        func_IP( input_packet );
        break;
case 0x0806:
        func_ARP( input_packet );
        break;
case 0x0001:
        func_IEEE8023( input_packet );
        break;
case 0x0002:
        ...
}
```

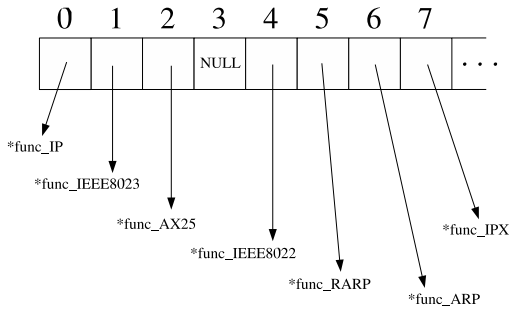
To je LINEARNO PRETRAŽIVANJE! Uz danu oznaku protokola, funkcija koja vrši obradu pronalazi se u vremenu O(n). Postoji bolje rješenje!

Zadnja znamenka može biti indeks u tablici funkcija. Funkcije za obradu okvira nazovimo na sljedeći način, gdje riječ PROTOCOL zamijenjana stvarnim nazivom protokola:

```
void func_PROTOCOL( struct packet *input_packet );
```

Struktura `packet` nije bitna. Ona predstavlja sadržaj primljenog paketa koji je potreno obraditi.

Napravimo slijednu listu (niz, array) od 16 pokazivača na navedene funkcije koja će izgledati kao što je prikazano na sljedećoj slici:



Funkcije `func_PROTOCOL` su tipa

```
typedef void (*proto_handler_func)(struct packet *ipacket);
```

Sad definiramo niz pokazivača na funkcije:

```
proto_handler_func proto_handler_list[16];
```

U fazi inicijalizacije (tj. pri pokretanju operacijskog sustava) podesimo pokazivače u nizu `proto_handler_list`:

```
proto_handler_list[0] = func_IP;
proto_handler_list[1] = func_IEEE8023;
proto_handler_list[2] = func_AX25;
proto_handler_list[3] = NULL;
proto_handler_list[4] = func_IEEE8022;
...
```

Uz ove definicije, obrada dolazećih paketa odvija se ovako:

```
...
proto_handler_func proto_func;
int index = proto_number % 16;
proto_func = proto_handler_list[ index ];
if ( proto_func != NULL )
        proto_func( input_packet );
else
        cannot_handle_packet();
...
```

Na ovaj način, složenost pretraživanja je bitno smanjena i praktično predstavlja najbrže moguće rješenje. Uz danu oznaku protokola, funkcija koja obrađuje paket pronalazi se u vremenu O(1).

Obratimo pažnju na liniju

```
int index = proto_number % 16;
```

Ova linija zapravo predstavlja preslikavanje $f : K \to M$ iz skupa protokolnih brojeva u skup memorijskih lokacija. Ovakvo preslikavanje je primjer preslikavanja koje se naziva *hash funkcija*. Općenito kažemo da hash funkcija preslikava skup ključeva u skup memorijskih lokacija.

Pri određivanju hash funkcije potrebno je prije svega odrediti njenu kodomenu, odnosno, raspon memorijskih lokacija u koje će se ključevi preslikavati. U navedenom primjeru, taj raspon sastoji se od 16 memorijskih lokacija numeriranih od 0 do 15. S obzirom da je broj ključeva jednak 10 (toliko ima podržanih protokola), neke od lokacija će ostati neiskorištene.

Omjer broja ključeva i broja memorijskih lokacija u koje ih hash funkcija preslikava naziva se *faktor opterećenja* λ. U primjeru, faktor opterećenja iznosi $\lambda = 10/16 = 0.625$. Što je faktor opterećenja bliži 1, to hash funkcija bolje iskorištava raspoloživi prostor.

Hash funkcija mora sama po sebi biti efikasna za proračun. Pretjerano složena hash funkcija u praksi usporava program iako će složenost cjelokupnog pretraživanja i dalje ostati O(1) dok god sama hash funkcija ne ovisi o broju ulaznih ključeva. U navedenom primjeru, usložnjavanje hash funkcije u cilju korištenja memorijskih lokacija 0–9 umjesto 0–15 u današnjim mjerilima nije potrebno.

Zadatak. Razvojem postojećeg operacijskog sustava, dodane su i funkcije za obradu paketa koji pripadaju sljedećim protokolima:

Protokolni brojevi uz dodatne protokole

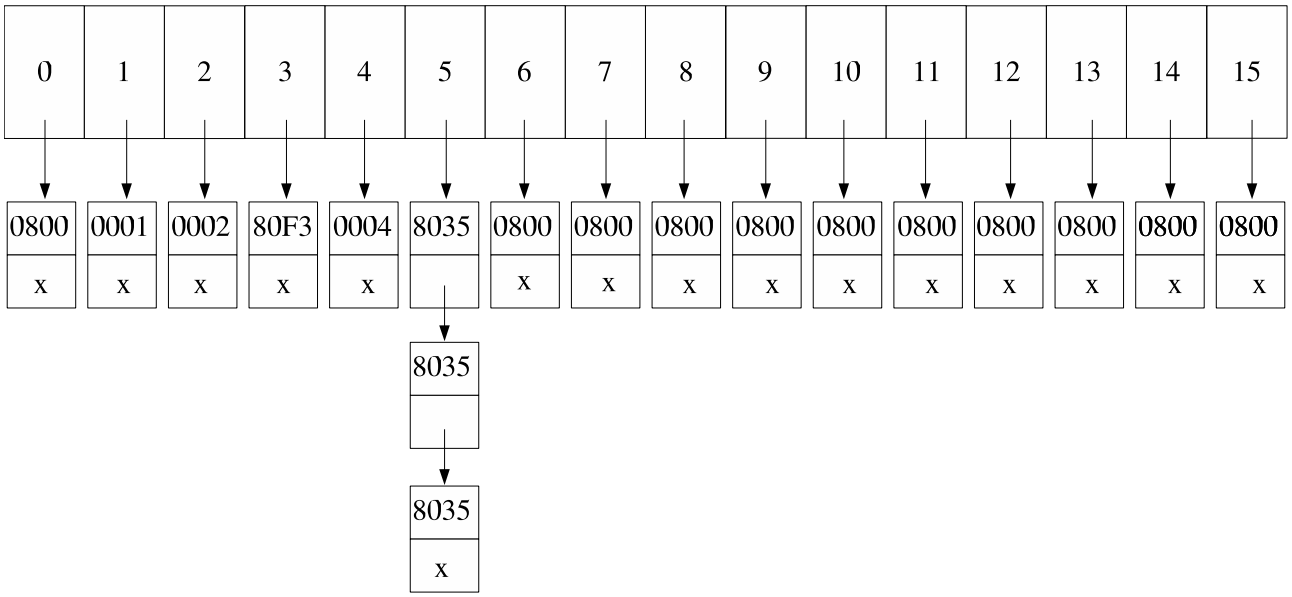
IP	0800
IEEE 802.3	0001
AX.25	0002
IEEE 802.2	0004
RARP	8035
ARP	0806
IPX	8137
Localtalk	0009
IPv6	86DD
BPQ	08FF
AppleTalk DDP	809B
AppleTalk AARP	80F3
X.25	0805
MOBITEX	0015

Za prve dvije funkcije postoji mjesto u tablici iz prethodnog zadatka tako da bi se podrška za AppleTalk lijepo uklopila u postojeći način rada. Međutim, funkcije za X.25 protokol i SNAP protokol bi se u nizu `proto_handler_list` morale nalaziti na istom mjestu kao i funkcija za RARP protokol. Naime, hash funkcija bi za sva tri protokola vraćala vrijednost 5. Ova pojava naziva se *kolizija*.

Postoji nekoliko načina za razrješavanje nastale kolizije. Mogli bismo, na primjer, povećati raspon korištenih memorijskih lokacija na 0–255 i upotrijebiti hash funkciju

```
int index = proto_num % 256;
```

Na taj način uklonili bi smo pojavu kolizije. Međutim, u tom slučaju, faktor opterećenja iznosio bi $\lambda = 14/256 \approx 0.055$ što bi bilo vrlo neefikasno korištenje memorijskih resursa. Drugi način rješavanja kolizije bio bi da se podaci ne smještaju direktno u memorijske lokacije koji vraća hash funkcija, nego da se na tim memorijskim lokacijama nalazi početak vezane liste u kojoj se nalaze svi podaci za koje hash funkcija vraća istu lokaciju. Primjer ovakve strukture za novi skup protokola izgledao bi kao što je prikazano na sljedećoj slici.



Napravimo strukturu:

```
struct packet_type {
  void (*func)( struct packet *input_packet );
  struct packet_type *next;
};

void func_IP ( struct packet *input_packet );
void func_ARP ( struct packet *input_packet );
void func_IEEE8023 ( struct packet *input_packet );

struct packet_type *ptype_base[16];
```

Zadatak. Za sljedeći skup ključeva, u programskom jeziku C napisati hash funkciju koja preslikava ključeve u memorijske lokacije od 0–7 bez kolizije:

```
0xB8345, 0x1FB24, 0xFF354, 0x34, 0xFF1A, 0x12375
```

Rješenje: Primijetimo da je druga zdesna znamenka za sve ključeve različita i da se kreće u rasponu od 0–7. To znači da hash funkcija mora za dani ključ vraćati vrijednost druge znamenke zdesna. Rješenje je dakle

```
int hash_func( int key )
{
  return (key & 0xF0) >> 4;  /* moguce je dodati i mod 8 na kraju */
}
```

Zadatak. Za sljedeći skup ključeva, u programskom jeziku C napisati hash funkciju koja preslikava ključeve u memorijske lokacije od 0–7 bez kolizije:

```
0xB8305, 0x1FB74, 0xFF324, 0xF4, 0xFFAA, 0x12355
```

Rješenje: Uočimo da druga znamenka po vrijednosti više ne može direktno biti upotrebljena kao vrijednost memorijske lokacije. Međutim, ukoliko njenu vrijednost uzimamo modulo 6, dobivamo brojeve u rasponu 0–5 bez kolizije. Stoga je hash funkcija

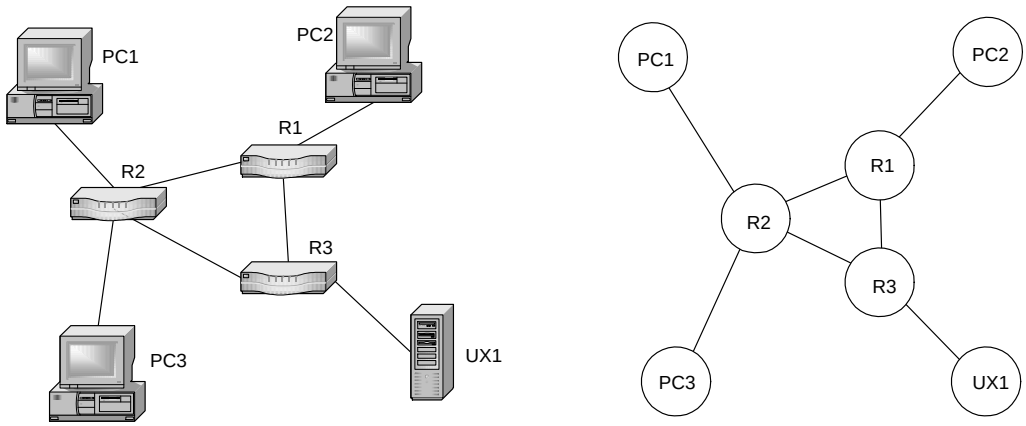
```
int hash_func( int key )
{
        return ((key & 0xF0) >> 4) % 6;
}
```

U navedenim primjerima skup ključeva bio je poznat i fiksan prije osmišljavanja hash funkcije. Međutim, ukoliko skup ključeva nije unaprijed zadan, nego ključevi dolaze slučajno iz nekog raspona brojeva, potrebno je voditi računa da hash funkcija što ravnomjernije raspoređuje ključeve po memorijskim lokacijama. S druge strane, ukoliko se očekuje da će se neki ključevi pojavljivati češće od drugih, poželjno je da hash funkcija takve ključeve smješta u lokacije sa što manje drugih ključeva.

Poglavlje 3

Grafovi

Mnoge se pojave i problemi u realnom svijetu daju opisati dijagramom koji se sastoji od skupa točaka povezanih linijama. Matematička struktura koja apstrahira ovakve dijagrame naziva se *graf*. Na slici 3.1 prikazana je jednostavna računalna mreža i njen model u obliku grafa.



Slika 3.1: Primjer mreže i njenog modela predstavljenog grafom

Definicija 1 Jednostavan graf je uređeni par (V, R) gdje je V konačan, neprazan skup, a R nerefleksivna, simetrična relacija na V.

Relacija R je nerefleksivna što znači da elementi iz V nisu u relaciji sami sa sobom (tj. $\forall a \in V, (a, a) \notin R$). Nadalje, relacija je simetrična što znači da ako je $(a, b) \in R$ onda je i $(b, a) \in R$.

Na primjer, jednostavan graf može biti zadan skupom

$$V = \{v_1, v_2, v_3, v_4\}$$

i relacijom

$$R = \{(v_1, v_2), (v_1, v_3), (v_2, v_1), (v_2, v_3), (v_3, v_1), (v_3, v_2), (v_3, v_4), (v_4, v_3)\}$$

Najčešće se sa E označava skup simetričnih parova[1] u R. U navedenom primjeru to znači da je

$$E = \Big\{ \{(v_1, v_2), (v_2, v_1)\}, \{(v_1, v_3), (v_3, v_1)\}, \{(v_2, v_3), (v_3, v_2)\}, \{(v_3, v_4), (v_4, v_3)\} \Big\}.$$

[1] Razlika između para i uređenog para je ta što je kod uređenog para točno određeno koji je prvi član para, a koji drugi, dok kod para to nije slučaj. To znači da su, na primjer, (a,b) i (b,a) isti par, ali ne i isti uređeni par.

Skup E naziva se skup grana u grafu.

Dakle, jednostavan graf G se sastoji od skupa čvorova V i skupa grana E, s tim da grane predstavljaju neuređene parove čvorova. Svaki graf, pa tako i jednostavan graf, uobičajeno se označava s $G = (V, E)$.

Ukoliko su grane predstavljene uređenim parom čvorova, dobiveni graf naziva se usmjereni graf.

Definicija 2 Usmjereni graf je uređeni par (V, R) gdje je V konačan, neprazan skup, a R nerefleksivna relacija na V.

Jedina razlika u odnosu na definiciju jednostavnog grafa sastoji se u činjenici da relacija R kod usmjerenog grafa ne mora biti simetrična.

Ova osnovna definicija grafa često se proširuje na način da se dozvole višestruke grane između istog para čvorova, kao i proširenje relacija R svojstvom refleksivnosti. S obzirom na ovo, moguće je nadalje proširiti definiciju grafa koja obuhvaća sva do sada navedena svojstva.

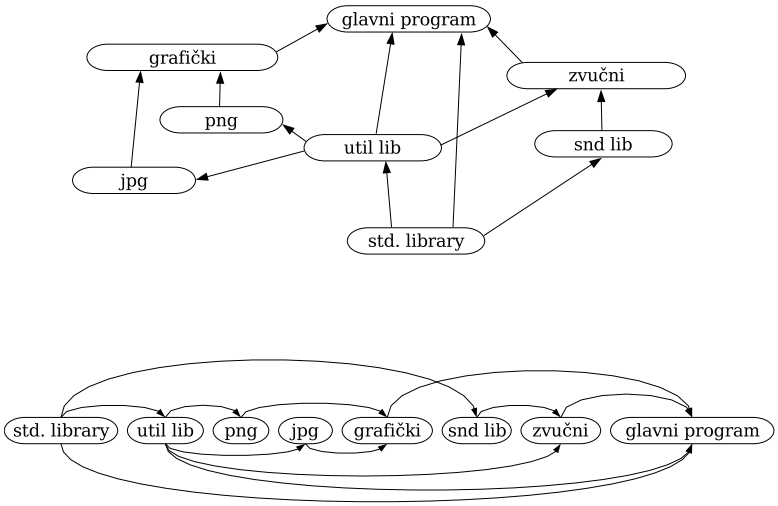
Definicija 3 Graf je uređeni par (V, E) gdje je V neprazan skup čvorova, a E je multiskup neuređenih parova čvorova.

U prethodnoj definiciji, E je multiskup koji, za razliku od skupa, može sadržavati više istih elemenata.

Slijede neka svojstva i pojmovi vezani uz grafove.

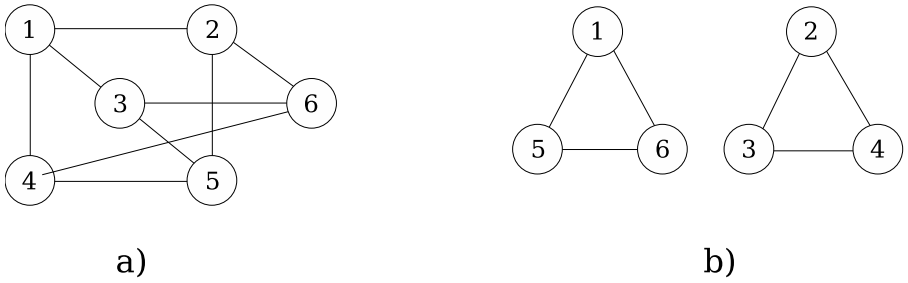
- **Red grafa** je broj njegovih čvorova, a **veličina grafa** je broj njegovih grana.
- Ako između čvorova u i v postoji grana e, kažemo da su čvorovi **incidentni** grani e kao i da je grana e **incidentna** čvorovima u i v.
- Dvije ili više grana koje povezuju isti par čvorova nazivaju se **paralelne grane**.
- Grana predstavljena parom čvorova u kojem čvorovi nisu međusobno različiti naziva se **petlja**.
- Graf koji ne sadrži petlje naziva se **multigraf**.
- Graf s barem jednom petljom naziva se **pseudograf**.
- Jednostavan graf je graf koji ne sadrži paralelne grane niti petlje.
- **Potpuni graf** K_n je jednostavan graf s n čvorova u kojem postoji točno jedna grana između svakog para različitih čvorova.
- Graf K_1 u kojem postoji samo jedan čvor i niti jedna grana naziva se **trivijalan graf**.
- **Bipartitni graf** je jednostavan graf u kojem se skup čvorova može podijeliti u dva skupa X i Y takva da svaka grana ima svojstvo da joj je jedan čvor u X, a drugi u Y. Ovakav graf označavamo s $G = (X, Y, E)$. (TODO Primjer matchings.)
- **Potpuni bipartitni graf** $K_{m,n}$ je jednostavan graf (X, Y, E) koji ima m čvorova u X i n čvorova u Y i u kojem postoji grana između svakog čvora iz X i svakog čvora iz Y.
- **Unija grafova** $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ je graf $G = G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$.
- **Usmjereni graf ili digraf** je graf kojeg čini skup čvorova V i skup *uređenih* parova međusobno različitih čvorova A nazvanih **lukovi**. (Vidjeti formalnu definiciju na početku ovog odjeljka.)

  Primjer usmjerenog grafa dan je na slici 3.2. Primjer pokazuje međusobnu ovisnost pojedinih dijelova neke programske cjeline. Grana od modula u prema modulu v označava da je modul u potrebno obraditi (tj. prevesti) prije modula v.
- Za luk (u, v) kažemo da je **usmjeren** od u prema v.
- Ako postoji luk (u, v) kažemo da je čvor v **susjedan** čvoru u. Također, kažemo da je čvor u **ulazno susjedan** čvoru v.



Slika 3.2: Primjer usmjerenog grafa i rezultata njegovog topološkog sortiranja

- **Ciklički graf** $C_n(V, E), n > 2$ je graf kod kojeg se čvorovi daju pobrojati tako da je $V = (1, 2, \ldots, n)$, a $E = \big((1, 2), (2, 3), \ldots, (n-1, n), (n, 1)$. Trokutasti graf je ciklički graf s tri čvora.
- **Komplement jednostavnog grafa** $G = (V, E)$ je jednostavan graf $\overline{G} = (V, F)$ u kojem grana između čvorova (u, v) postoji ako i samo ako ta grana u G ne postoji (slika 3.3).



Slika 3.3: Graf i njegov komplement

3.1 Izomorfizam grafova

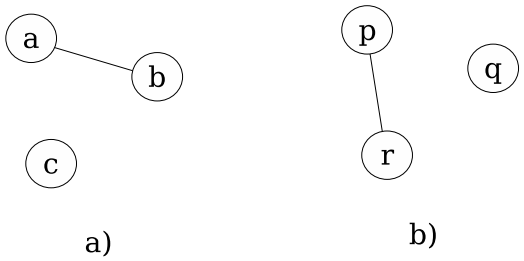
Kažemo da su dva grafa $G_1 = (V, E)$ i $G_2 = (V', E')$ **identična** ako i samo ako je $V = V'$ i $E = E'$. U praksi je ovakvo uspoređivanje često prestrogo. Na primjer, grafovi na slici 3.4 nisu identični iako je jasno da posjeduju vrlo veliku sličnost te s obzirom na primjenu predstavljaju iste grafove. Stoga se za usporedbu grafova definira pojam izomorfnosti grafova na sljedeći način.

Definicija 4 Za grafove $G = (V, E)$ i $G' = (V', E')$ kažemo da su **izomorfni** ako postoji bijektivno preslikavanje $f : V \to V'$ takvo da je $\big(f(v), f(w)\big)$ grana u G' ako i samo ako je (v, w) grana u G.

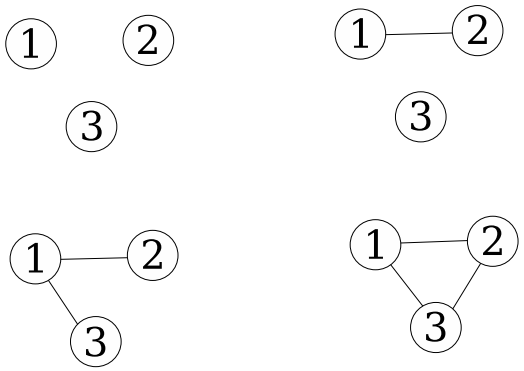
U praktičnoj primjeni, izomorfnost dvaju grafova dovoljna je da ih se tretira kao isti graf.

Ukoliko sve međusobno izomorfne grafove smatramo jednim te istim grafom, zaključujemo sljedeće činjenice.

- Postoji samo jedan jednostavan graf *reda* jedan i to je *trivijalan graf* (tj. graf sa samo jednim čvorom).
- Postoje dva neizomorfna jednostavna grafa reda dva.



Slika 3.4: Grafovi a) i b) nisu identični

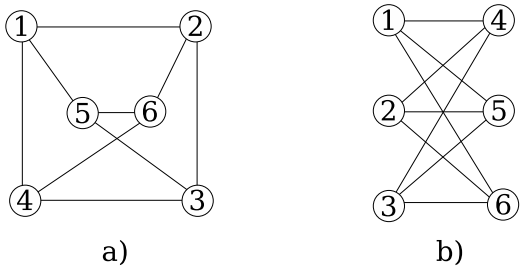


Slika 3.5: Svi neizomorfni grafovi reda 3

- Postoje četiri jednostavna grafa reda 3 (slika 3.5).

Primjer 3.1 Jesu li grafovi na slici 3.6 izomorfni?

Rješenje: Da. $f(1) = 1, f(2) = 4, f(3) = 3, f(4) = 6, f(5) = 5, f(6) = 2$. Oba su izomorfna s $K_{3,3}$.



Slika 3.6: Grafovi uz primjer 3.1

Za dva zadana grafa koji imaju jednak red i veličinu problem određivanja da li postoji izomorfizam između njih je poznat pod nazivom **problem izomorfizma grafova**. Složenost postojećih algoritama koji rješavaju ovaj problem je eksponencijalna.

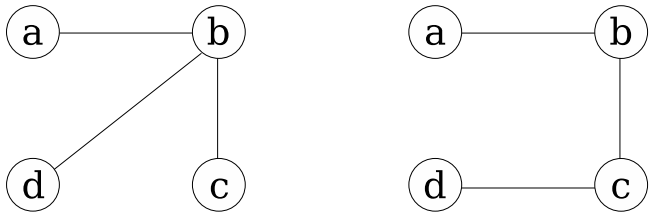
Primjer 3.2 Pokažite da dva grafa istog reda i veličine ne moraju biti izomorfni.

Rješenje: Vidljivo iz grafa na slici 3.7.

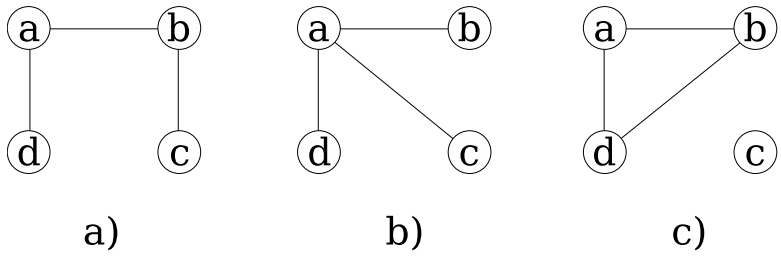
Jednostavan graf koji je izomorfan svom komplementarnom grafu naziva se **samo-komplementarni** graf.

Primjer 3.3 Utvrdite koji od grafova na slici 3.8 su samo-komplementarni.

Rješenje: Graf a) je samo-komplementaran, dok grafovi b) i c) to nisu. Komplement od b) je izomorfan s c) i obratno.



Slika 3.7: Grafovi istog reda i veličine ne moraju biti izomorfni



Slika 3.8: Slika uz primjer 3.3

3.2 Podgraf

Za graf $H = (W, F)$ kažemo da je **podgraf** od grafa $G = (V, E)$ ako je W podskup of V, a F podskup od E.

- Ako je podgraf H grafa G ciklički graf, H se naziva **ciklus** u G.
- Potpuni podgraf od G se naziva **klika** u G.
- Graf G' kojem je graf G podgraf naziva se **nadgraf** od G.
- Svaki podgraf $H = (V, F)$ od $G = (V, E)$ naziva se **razapinjući podgraf** od G. To je stoga što uključuje sve čvorove.

Ako je $F \subseteq E$ u grafu $G = (V, E)$, razapinjući podgraf dobiven brisanjem grana iz F u grafu G označava se s $G - F$. Analogno, za čvorove W, pišemo $G - W$.

- Ako je $H = (W, F)$ podgraf of $G = (V, E)$ takav da u F postoji grana između dvaju čvorova iz W ako i samo ako ista grana postoji i u E, za podgraf H kažemo da je **induciran skupom** W. Takav podgraf H označavamo s $\langle W \rangle$. To je ujedno i maksimalni podgraf od G s obzirom na skup W.

3.3 Stupanj čvora, ulazni stupanj, izlazni stupanj

Ukoliko u nekom čvoru v postoji p petlji i q grana (ne petlji), **stupanj čvora** v iznosi $2p + q$. U grafu bez petlji, stupanj čvora je broj grana koji su incidentne tom čvoru. Također, čvor koji ima stupanj 0 nazivamo **izoliranim čvorom** dok čvor koji ima stupanj 1 nazivamo **krajnjim čvorom**.

Maksimalni stupanj od svih čvorova u grafu G označavamo s $\Delta(G)$, dok minimalni stupanj označavamo s $\delta(G)$.

Graf nazivam **k-regularnim** ukoliko je stupanj svakog čvora jednak k. Graf nazivamo **regularnim** ukoliko postoji nenegativan cijeli broj k takav da je graf k-regularan.

Pri zbrajanju stupnjeva svih čvorova u grafu, svaka grana se ubraja dvaput, tj. svaka grana uvećava za po 1 stupnjeve dvaju čvorova koje povezuje. Sljedeći teorem je poznat kao prvi teorem iz teorije grafova.

Teorem 3.1 (Euler) Zbroj stupnjeva svih čvorova jednak je dvostrukom broju grana u grafu.

Čvor nazivamo **parnim** ukoliko mu je stupanj paran broj, a **neparnim** u suprotnom.

Teorem 3.2 Svaki graf ima paran broj neparnih čvorova.

Dokaz. Označimo s x zbroj stupnjeva neparnih čvorova, a s y zbroj stupnjeva parnih čvorova. Broj y je paran. Broj $x + y$ je također paran jer je jednak dvostrukom broju grana. Slijedi da je i broj x paran. Ako je u grafu p neparnih čvorova, paran broj x predstavlja zbroj od p neparnih brojeva. Zaključak: p mora biti paran. □

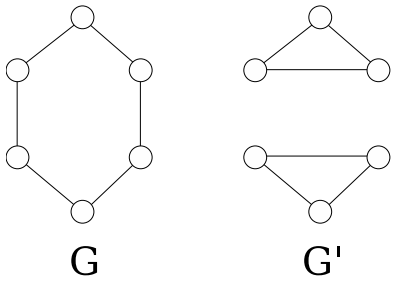
Primjer 3.4 Pronađite broj grana u potpunom bipartitnom grafu $K_{m,n}$.

Rješenje: Neka je $K_{m,n} = (X, Y, E)$. Svakom čvoru iz X incidentno je n grana (jedna prema svakom čvoru iz Y). S obzirom da postoji m čvorova u X, ukupan broj grana iznosi mn.

U usmjerenom grafu, broj grana koje izlaze iz nekog čvora naziva se **izlazni stupanj** čvora, dok se broj grana koje ulaze u čvor naziva **ulazni stupanj** čvora. Zbroj izlaznih stupnjeva svih čvorova u grafu jednak je broju lukova u grafu. Ovaj zbroj je također jednak i zbroju svih ulaznih stupnjeva čvorova u grafu.

Primjer 3.5 Pokažite da dva grafa sa istim skupom čvorova V u kojem svaki čvor i iz jednog grafa ima isti stupanj kao čvor i iz drugog, ne moraju nužno biti izomorfni.

Rješenje: Slika 3.9.



Slika 3.9: Slika uz primjer 3.5

3.4 Zapis grafova u računalu

U ovom odjeljku opisani su neki od načina zapisivanja grafova u računalu.

3.4.1 Matrice susjednosti i incidencije

Neka je $G = (V, E)$ graf u kojem je $V = \{1, 2, \ldots, n\}$. **Matrica susjednosti** grafa G je $n \times n$ matrica $A = [a_{ij}]$ gdje je nedijagonalni element a_{ij} jednak broju grana koje spajaju čvorove i i j, a dijagonalni element a_{ii} jednak dvostrukom broju petlji u čvoru i. Matrica ovakvog grafa je simetrična.

Matrica susjednosti za jednostavan graf je binarna matrica, tj. svi elementi su joj 0 ili 1.

Matrica susjednosti usmjerenog grafa s čvorovima $V = \{1, 2, \ldots, n\}$ je $n \times n$ matrica $A = [a_{ij}]$ gdje je $a_{ij} = 1$ ako i samo ako postoji luk od i do j. Dijagonalni elementi iznose 0 i matrica nije nužno simetrična.

Neka je $G = (V, E)$ jednostavan graf u kojem je $V = \{1, 2, \ldots, n\}$ i $E = (e_1, e_2, \ldots, e_m)$. **Matrica incidencije** $B = [b_{ij}]$ grafa G se definira na sljedeći način. Redak i matrice B odgovara čvoru i za svaki čvor i. Stupac k odgovara grani e_k. Ako je e_k grana grana koja povezuje čvorove i i j, elementi b_{ik} i b_jk matrice iznose 1, dok su svi drugi elementi u tom stupcu jednaki 0. Ako je G digraf, a e_k je luk od i do j, definiramo da je $b_{ik} = -1$, $b_{jk} = 1$, a svi ostali elementi u stupcu k su 0.

Primijetimo da za matricu susjednosti i matricu incidencije vrijede sljedeći teoremi.

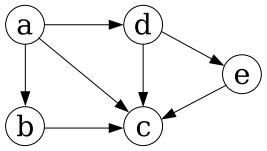
Teorem 3.3 U matrici susjednosti grafa, suma elemenata u retku (ili stupcu) koji odgovara nekom čvoru jednaka je stupnju tog čvora, dok je suma svih elemenata u matrici jednaka dvostrukom broju grana u grafu.

Teorem 3.4 U matrici susjednosti usmjerenog grafa, suma elemenata u retku koji odgovara nekom čvoru jednaka je izlaznom stupnju tog čvora, dok je suma elemenata u stupcu jednaka ulaznom stupnju čvora koji odgovara tom stupcu. Suma svih elemenata matrice jednaka je broju lukova u grafu.

3.4.2 Lista susjednosti

Zapis grafa u obliku matrice susjednosti posjeduje niz nedostataka. Na primjer, dodavanje novog čvora u graf ili brisanje čvorova iz grafa s ovakvom reprezentacijom predstavlja složen zadatak. Nadalje, u praksi se vrlo često pojavljuju grafovi kod kojih je broj grana malen u usporedbi s maksimalno mogućim brojem grana. To znači da je u matrici susjednosti velik broj elemenata jednak 0 te da opis tih nepostojećih grana nepotrebno zauzima memorijski prostor.

Stoga se struktura graf uobičajeno implementira na sljedeći način. Razmotrimo primjer grafa na slici 3.10.



čvor	lista susjeda
a	b,c,d
b	c
c	-
d	c,e
e	c

Slika 3.10: Primjer grafa i pripadajuće liste susjednosti

3.5 Putovi i ciklusi u grafu

Često se u vezi s grafovima postavlja pitanje može li se korištenjem postojećih grana u grafu dospijeti od jednog čvora u grafu do nekog drugog ili pak do svih ostalih čvorova. Ovo pitanje naziva se pitanje povezanosti te će o njemu i srodnim pojmovima biti riječi u ovom i sljedećih nekoliko odjeljaka.

Neka su v i w dva čvora u grafu. **Šetnja od** v **do** w u grafu je konačna, alternirajuća sekvenca čvorova i grana $v = v_0, e_1, v_1, e_2, v_2, e_3, \ldots, e_n, v_n = w$ takva da svaka grana e_i u sekvenci spaja čvorove v_{i-1} i v_i. Čvorovi u grane koje čine šetnju ne moraju biti međusobno različiti.

Dvije šetnje $v_0, e_1, v_1, \ldots, e_n, v_n$ i $u_0, f_1, u_1, \ldots, f_m, u_m$ su **jednake šetnje** ako je $n = m$, $v_i = u_i$ i $e_i = f_i$, $0 \leq i \leq n$. Broj grana u šetnji naziva se **duljina šetnje**.

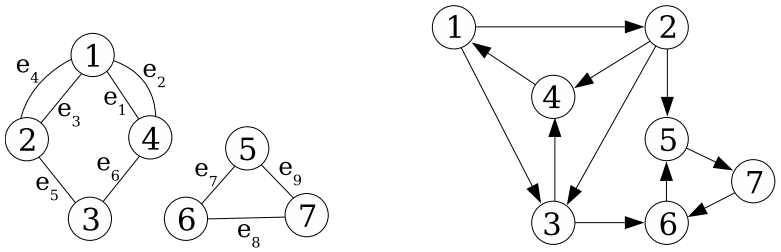
Ako je graf jednostavan, nije potrebno pisati eksplicitno grane već je dovoljno navesti čvorove. Na primjer, šetnja $v_0, e_1, v_1, e_2, v_2 \ldots, e_n, v_n$ se može izraziti kao $v_0, v_1, \ldots, v_n$. Šetnja u kojoj nema ponovljenih grana naziva se **trag**. Šetnja od v do w u kojoj nema ponovljenih čvorova naziva se **put** od v do w. Primijetite da je svaki put ujedno i trag.

Ako su v i w čvorovi u usmjerenom grafu, **usmjerena šetnja** od v do w je konačna sekvenca $v = v_0, a_1, v_1, a_2, v_2, \ldots, a_n, v_n = w$ takva da je svaki luk a_i u sekvenci luk između čvorova v_{i-1} i v_i. Analogno su definirani i pojmovi **usmjereni trag** i **usmjereni put**.

Primjer 3.6 Na slici 3.11-a), sekvenca $2, e_4, 1, e_1, 4, e_2, 1, e_1, 4$ je šetnja od čvora 2 do čvora 4. Sekvenca $2, e_3, 1, e_1, 4, e_2, 1, e_4, 2, e_5, 3$ je trag između 2 i 3. Sekvenca $2, e_5, 3, e_6, 4, e_1, 1$ je put između 2 i 1. Nadalje, na slici 3.11-b), $2, 3, 4, 1, 2, 3, 6$ je usmjerena šetnja od 2 do 6, sekvenca $2, 4, 1, 3$ je usmjereni trag od 2 do 3, dok je $1, 2, 5, 7$ usmjereni put od 1 do 7.

Teorem 3.5 Svaka šetnja u grafu od čvora v do w sadrži put od v do w.

Analogna tvrdnja vrijedi za usmjerenu šetnju i usmjereni put.



Slika 3.11:

Teorem 3.6 Ako je A matrica susjednosti jednostavnog grafa $G = (V, E)$, gdje je $V = \{1, 2, \ldots, n\}$, a_{ij} element u matrici A^k jednak je broju različitih šetnji duljine k od čvora i do j.

Dokaz. Indukcijom. Za $k = 1$ vrijedi. Pretpostavka indukcije: vrijedi za $(k-1)$. $A^k = A^{k-1} \cdot A$. a_{ij} je 1 ako i samo ako su i i j susjedi. To znači da do susjeda ima jedna šetnja duljine jedan, a od susjeda ima $A^{k-1}(i, j)$ šetnji duljine $k-1$. □

Primjer 3.7 Što predstavljaju elementi na dijagonali matrice A^2?

Rješenje: Svaki element a_{ii} predstavlja stupanj čvora i.

Zatvorena šetnja u grafu je šetnja od čvora do sebe samoga. Zatvorena šetnja bez ponovljenih grana naziva **kolo**. Kolo bez ponovljenih čvorova osim početnog naziva se **ciklus**. Podgraf C jednostavnog grafa G je **ciklus u G** ako i samo ako je C ciklički graf.

U jednostavnom grafu G, svaki ciklus od k čvorova naziva se k-ciklus. S obzirom na to da li je broj k paran ili neparan, k-ciklus također nazivamo parnim ili neparnim. Kod usmjerenih grafova, pojmovi usmjereno kolo i usmjereni ciklus definirani su analogno. Koristeći pojam ciklusa, sljedeći teorem izriče nužan i dovoljan uvjet da bi graf bio bipartitan.

Teorem 3.7 Jednostavan graf s tri ili više čvorova je bipartitan ako i samo ako ne sadrži neparne cikluse.

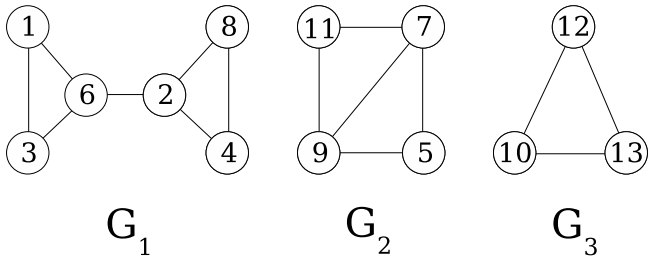
3.6 Povezanost

Za par čvorova u grafu kažemo da su **povezani** ako između njih postoji put. Za graf kažemo da je **povezan** ako su svaka dva čvora u njemu povezani; u suprotnom graf nazivamo nepovezanim. Povezani podgraf H grafa G zovemo **komponentom od** G ako H nije sadržan niti u jednom povezanom podgrafu grafa G koji ima više čvorova ili grana nego H. Drugim riječima, maksimalni povezani podgrafi nekog grafa čine komponente tog grafa. Graf je povezan ako i samo ako je broj komponenata u njemu jednak jedan.

Ako je $G = (V, E)$, graf u kojem su iz skupa grana uklonjene grane koje pripadaju nekom skupu F označavamo s $(V, E \setminus F)$ ili s $G - F$, a ako F sadrži samo jednu granu f, onda pišemo $G - f$.

Skup $F \subset E$ povezanog grafa G nazivamo **odspajajućim skupom** u G ako je broj komponenti grafa $(V, E \setminus F)$ strogo veći od broja komponenti grafa G. Ako se odspajajući skup sastoji samo od jedne grane f, ta se grana naziva **most**. Graf nazivamo **k-povezanim s obzirom na grane** ako svaki odspajajući skup ima najmanje k grana. **Stupanj povezanosti s obzirom na grane** $\lambda(G)$ grafa G predstavlja minimalan broj grana koje je potrebno ukloniti iz grafa da bi graf postao nepovezan. Po definiciji, $\lambda(G) = 0$ kad je $G = K_1$. Odspajajući skup F nazivamo **rezom** ukoliko niti jedan pravi podskup od F nije odspajajući skup.

Primjer 3.8 Na slici 3.12 prikazan je graf s 13 čvorova. Taj graf nije povezan jer postoje parovi čvorova koji nisu povezani (npr. čvorovi 7 i 12). Komponente grafa su podgrafovi G_1, G_2 i G_3. Skup $\{(1, 6), (2, 6)\}$ je odspajajući skup, ali ne i rez jer je njegov podskup $\{(2, 6)\}$ također odspajajući skup. Grana $(2, 6)$ je most.



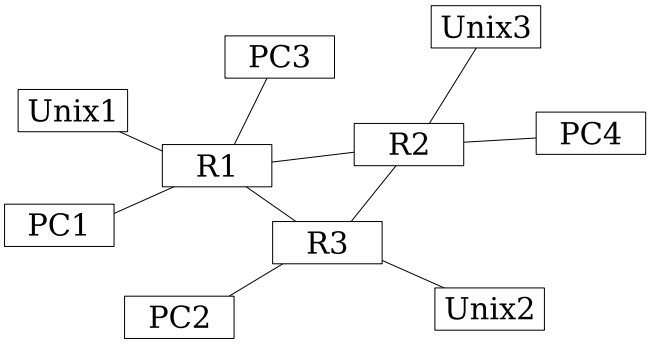
Slika 3.12: Graf od tri komponente (uz primjer 3.8)

Svaka grana koja čini dio nekog ciklusa u G može biti uklonjena iz G bez straha da će to izazvati odspajanje grafa. Sljedeći teorem ovo formalizira ustanovljavajući vezu između mosta i ciklusa.

Teorem 3.8 Neka je G povezan graf. Grana e u grafu G čini most ako i samo ako e nije sadržana niti u jednom ciklusu u G.

Primjer 3.9 Za računalnu mrežu prikazanu na slici 3.13 odredite da li postoji kritična grana, tj. grana koja svojim ispadom uzrokuje nepovezanost mreže.

Rješenje: Potrebno je, dakle, odrediti barem jednu granu koja čini most. Svaka od grana koje povezuju krajnja računala na usmjerivače predstavlja most. U slučaju da se neko računalo smatra važnim, ono je moguće vezati na dva usmjerivača te se na taj način osigurava da ispad jedne od njegovih grana ne uzrokuje njegovo odspajanje od ostatka mreže. To bi, na primjer, značilo da je računalo Unix2 spojeno i na usmjerivač R3, ali i na usmjerivač R2. Ovakva topologija često se naziva engl. *dual homing*.



Slika 3.13: uz primjer 3.9

Primjer 3.10 Za sustav cesta u nekom gradu (TODO:nema još slike), odredite rez najmanje veličine.

Prethodni pojmovi odnosili su se na grane. Analogni koncepti postoje i za čvorove. Za početak uvedimo sljedeću konvenciju zapisivanja. Ako je $G = (V, E)$ graf i W neki podskup $W \subseteq V$, onda sa $G - W$ označavamo graf dobiven uklanjanjem čvorova navedenih u W iz grafa G kao i svih grana incidentnih tim čvorovima. Ukoliko se W sastoji od jednog elementa w, skraćeno pišemo $G - w$.

Skup W čvorova iz G nazivamo **razdvajajućim skupom** ako je broj komponenti grafa $G - W$ strogo veći od broja komponenti grafa G. Ako se razdvajajući skup sastoji od jednog čvora, taj čvor nazivamo **čvor reza**.

Primjer 3.11 Odredite jedan razdvajajući skup u grafu K_5.

Rješenje: Nema ga. U kompletnom grafu ne postoji razdvajajući skup.

Stupanj povezanosti $\kappa(G)$ grafa G predstavlja minimalni broj čvorova čijim uklanjanjem iz grafa, graf postaje ili nepovezan ili trivijalan (tj. spada na samo jedan čvor). Graf nazivamo **k-povezanim** ako je $\kappa(G) \geq k$.

U povezanim grafovima definiramo sljedeće veličine vezane uz putove. Broj grana na putu između dvaju čvorova v i w koji ima najmanje grana nazivamo najkraći put i označavamo ga s $d(v,w)$. **Ekscentricitet** $e(v)$ čvora v je maksimalna vrijednost $d(v,w)$ od v do svih ostalih čvorova. **Polumjer** $r(G)$ grafa G je ekscentricitet čvora s najmanjim ekscentricitetom. Čvor v nazivamo **središnjim čvorom** ako mu je ekscentricitet jednak polumjeru grafa. **Središte grafa** $C(G)$ je skup svih središnjih čvorova.

Primjer 3.12 Internet, traceroute.

Lokacija	broj skokova
srce.hr	4
Google	18
Microsoft.com	19
Linux.org	25

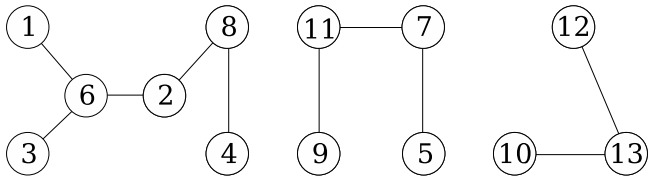
Kod usmjerenih grafova, pojmovi vezani uz povezanost su nešto drukčije definirani. Za usmjereni graf G kažemo da je **strogo povezan** ako postoji usmjereni put između svakog para čvorova. **Stroga komponenta** usmjerenog grafa je maksimalni strogo povezani podgraf. Usmjereni graf je **unilateralno** povezan ako za svaki par čvorova v i w postoji bilo usmjereni put od v do w ili usmjereni put od w do v.

3.7 Stabla i razapinjuća stabla

Aciklički graf ili šuma je graf u kojem nema ciklusa. **Stablo** je povezani aciklički graf. Dakle, svaka komponenta šume je stablo i svako stablo je povezana šuma.

Primjer 3.13 Na slici 3.14 prikazan je graf. Odredite stabla u tom grafu. Je li taj graf šuma?

Rješenje: Pazi! Ima puno stabala. Na primjer, {(1,6),(2,6)} je jedno od stabala. Dakle potrebno je raditi razliku između komponente (maksimalni povezani podgraf) i stabla u ovakvom grafu.



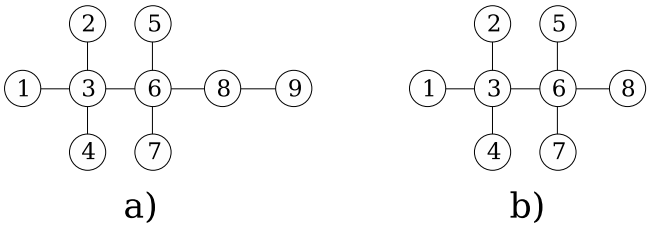
Slika 3.14: uz primjer 3.13

Teorem 3.9 Sljedeće tvrdnje su ekvivalentne u grafu G s n čvorova:

- G je stablo.
- Postoji *jedinstven* put između svaka dva čvora u G.
- G je povezan i svaka grana u G je most.
- G je povezan i ima $(n-1)$ granu.
- G je acikličan i ima $(n-1)$ granu.
- G je acikličan i kad se bilo koja dva nesusjedna čvora u G povežu granom, rezultirajući graf posjeduje jedinstveni ciklus.
- G je povezan, i kad se bilo koja dva nesusjedna čvora u G povežu granom, rezultirajući graf posjeduje jedinstveni ciklus.

Primjer 3.14 Odredite središta stabala prikazanih na slikama 3.15-a) i b).

Rješenje: Za graf na slici a) središte je čvor 6, dok je za b) središte skup $\{3,6\}$.



Slika 3.15: uz primjer 3.14

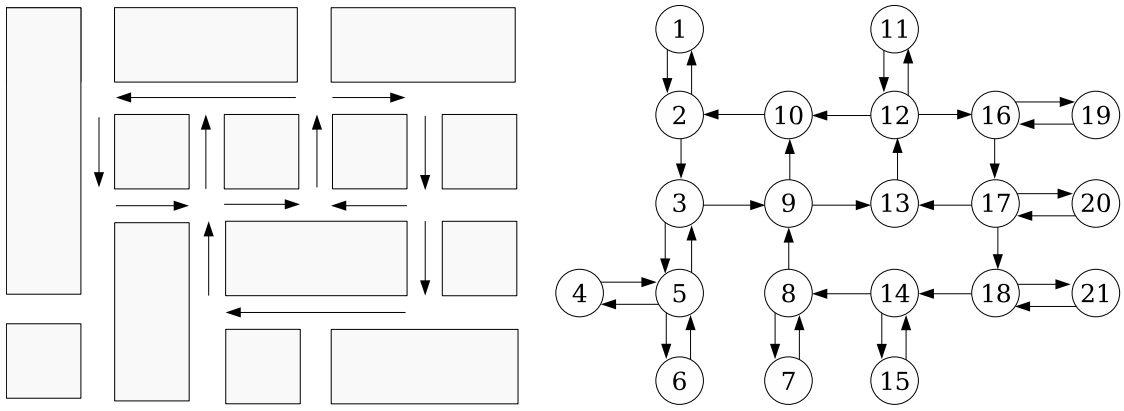
Aciklički povezani razapinjući podgraf nazivamo **razapinjuće stablo**. Dakle, to je stablo u grafu koje obuhvaća sve čvorove tog grafa.

Teorem 3.10 Graf je povezan ako i samo ako posjeduje razapinjuće stablo.

Dokaz. Neka je G povezan graf. Provodeći postupak brisanja grana koje nisu mostovi iz G, dobivamo povezani podgraf H u kojem je svaka grana most. Tada je H razapinjuće stablo. U suprotnom smjeru, ako postoji razapinjuće stablo u G, onda postoji put između svakog para čvorova u G pa je G povezan.

Primjer 3.15 U nekom gradu, gradska uprava odlučila je određeni broj cesti pretvoriti u jednosmjerne. Raspored cesti i njihova orijentacija prikazani su na slici 3.16. Ukoliko cesta nema strelicu koja označava orijentaciju, to znači da je cesta dvosmjerna. Utvrdite hoće li novi sustav cesti osigurati da se iz svake točke u gradu može doći do svake druge točke.

Rješenje: Ako sustav cesti predstavimo usmjerenim grafom u kojem su raskrižja cesti predstavljena čvorovima, a ceste su predstavljene usmjerenim granama, s tim da je svaka dvosmjerna cesta predstavljena dvjema suprotno usmjerenim granama, potrebno je utvrditi da li je takav graf strogo povezan.



Slika 3.16: Ceste u gradu (skica i graf, uz primjer 3.15 o povezanosti grafa)

Algoritam koji bi utvrđivao da li je proizvoljni usmjereni graf strogo povezan ili ne, mogao bi djelovati na sljedeći način. Uzmemo jedan proizvoljni čvor za koji ćemo utvrditi da li se iz njega može doći do svih ostalih čvorova. Prije svega, sâm početni čvor označimo kao dostupan jer je svakako dostupan od samog sebe. Zatim, počevši od tog čvora, *zapamtimo* sve susjede do kojih se iz njega može direktno doći i označimo te čvorove kao dostupne. Uzmemo jedan od *zapamćenih* čvorova i ponovimo prethodni postupak, tj. od tog čvora zapamtimo sve nove još nezapamćene susjede za koje još nismo utvrdili dostupnost i označimo ih dostupnima. Kad više nema zapamćenih čvorova, broj čvorova koji su u postupku obilježeni kao dostupni mora odgovarati broju čvorova u grafu.

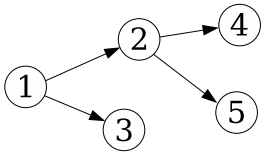
Kad smo za konkretni početni čvor utvrdili da su svi ostali čvorovi dostupni iz njega, potrebno je utvrditi da li je iz svih ostalih čvorova moguće doći do tog čvora. Ako se iz svih ostalih čvorova može doći do tog čvora, usmjereni graf je povezan. Zašto? Zato jer između svaka dva čvora postoji put koji vodi preko čvora od kojeg je izvršen algoritam. Algoritam koji od jednog čvora prolazi do svih ostalih čvorova poštujući njihovu povezanost

granama nazivamo *prolazak kroz graf*. Da bismo utvrdili da li se iz svih ostalih čvorova može dospijeti do nekog određenog čvora prolazak možemo prethodno opisani prolazak kroz graf modificirati na način da su sljedeći posjećeni susjedi oni od kojih postoji grana *prema* izvorišnom čvoru, umjesto *od* izvorišnog čvora. Na taj način dobivamo putove od svih ostalih čvorova do nekog odredišnog čvora.

U prethodnom opisu algoritma koristili smo glagol *zapamtiti*. Pri konkretnoj realizaciji algoritma, struktura koju ćemo koristiti da bismo pamtili čvorove bit će stog. Dakle, kad smo govorili o pamćenju nekog čvora, to je zapravo značilo pohranjivanje čvora u stog[2].

Algoritam 3.7.1 predstavlja općenitu verziju tzv. prolaska kroz usmjereni graf *u dubinu* (engl. *depth first search*). Ovaj izraz *u dubinu* slijedi iz činjenice da algoritam obrađuje čvorove na način da prodire sve dublje i dublje u graf (tj. dalje i dalje od početnog čvora) te se, kad dođe do maksimalne dubine, vraća na „manje duboke" čvorove i nastavlja obradu preostalih čvorova na isti način.

Algoritam 3.7.1 na izlazu daje tzv. DFS šumu. Ova šuma zapisana je pomoću niza koji svakom čvoru dodijeljuje njegovog prethodnika u odgovarajućem stablu. Jedan primjer takvog niza i njemu pripadajućeg stabla prikazan je na slici 3.17.



v	1	2	3	4	5
pred[v]	NULL	1	1	2	2

Slika 3.17: Stablo i pripadajući niz prethodnika

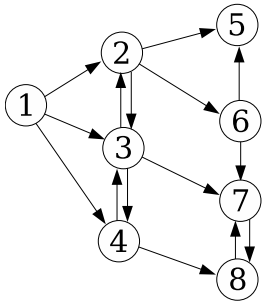
Ova šuma predstavljena je grafom $G_\pi(V, E_\pi)$ gdje je $E_\pi = \{(\pi[v], v) : v \in V \text{ i } \pi[v] \neq 0\}$.

Vratimo se sada na primjer utvrđivanja povezanosti cesta u gradu. Problem možemo riješiti na način da provedemo DFS prolazak kroz graf od svakog čvora do svih ostalih. Ukoliko je od svakog čvora moguće doći do svih ostalih, to upravo znači da je usmjereni graf strogo povezan.

Kako bi se promijenio obilazak čvorova grafa ukoliko bismo umjesto stoga za pamćenje čvorova koristili rep? Korištenje repa rezultira u bitno drugačijem prolasku kroz graf. Ovakav prolazak naziva se **prolazak u širinu** (engl. *Breadth First Search*, BFS). Slijedi primjer DFS i BFS prolazaka kroz usmjereni graf.

Primjer 3.16 Za usmjereni graf na slici 3.18 odredite:

1. redoslijed posjećivanja čvorova pri DFS prolasku kroz graf,
2. redoslijed posjećivanja čvorova pri BFS prolasku kroz graf.



Slika 3.18: Primjer grafa za prolaske kroz graf

U sljedeće dvije tablice prikazani su DFS i BFS prolasci uz navedene čvorove i stanje stoga, odnosno repa u svakom koraku.

[2]U nastavku teksta vidjet ćemo da umjesto stoga možemo koristiti i druge strukture kao, na primjer, rep te da se uslijed toga bitno mijenja ponašanje algoritma

algoritam 3.7.1 DFS prolazak kroz usmjereni graf

Ulaz: usmjereni graf $G = (V, E)$

Izlaz: podgraf prethodnika $G_\pi(V, E_\pi)$, gdje je $E_\pi = \{(\pi[v], v) : v \in V \text{ i } \pi[v] \neq 0\}$

Lokalne varijable: S stog, $M[v] = \begin{cases} 0 & \text{netaknut} \\ 1 & \text{u stogu} \\ 2 & \text{obrađen} \end{cases}$

```
1: procedure DFS(G)
2: for svaki čvor u ∈ V(G) do
3:    M[u] ← 0
4:    π[u] ← NULL
5: end for
6: for all u ∈ V(G) do
7:    if M[u] = 0 then
8:       call DFS_visit(G,u)
9:    end if
10: end for
11: end proc
12:
13: procedure DFS_visit(G,v)
14: Push(S,v)
15: M[v] ← 1
16: while not Empty(S) do
17:    w ← Pop(S)
18:    Visit(w) {Obradi čvor}
19:    M[w] ← 2
20:    for all s susjedan čvoru w do
21:       if M[s] = 0 then
22:          π[s] ← w
23:          Push(S,s)
24:          M[s] ← 1
25:       end if
26:    end for
27: end while
```

BFS prolazak

Trenutni čvor	Elementi u repu
1	2,3,4
2	3,4,5,6
3	4,5,6,7
4	5,6,7,8
5	6,7,8
6	7,8
7	8
8	$\emptyset$

DFS prolazak

Trenutni čvor	Elementi u stogu
1	4,3,2
4	8,3,2
8	7,3,2
7	3,2
3	2
2	6,5
6	5
5	$\emptyset$

Primijetimo sljedeće zanimljivo svojstvo BFS algoritma. Čvorovi se posjećuju redoslijedom koji određuju njihove udaljenosti (tj. najmanji broj grana) od početnog čvora i to na način da se prvo posjete čvorovi koji su od početnog udaljeni jednu granu, zatim čvorovi koji su udaljeni dvije grane i tako dalje. Upravo iz ove činjenice slijedi i sam naziv algoritma *pretraživanje u širinu*. To znači da se s obzirom na udaljenost od početnog čvora, čvorovi obrađuju u „koncentričnim kružnicama".

S obzirom na navedeno svojstvo BFS algoritma, taj algoritam moguće je koristiti i za pronalaženje najkraćih puteva od nekog čvora do svih ostalih, u slučaju kad je duljina puta definirana brojem grana na putu. Pri izvođenju algoritma, u trenutku kad se svi susjedi nekog čvora stavljaju u rep, poznato je da je udaljenost do

svakog od tih susjeda za jedan veća od udaljenosti čvora koji se trenutno obrađuje, tj. čvora koji njima prethodi. Na taj način moguće je odrediti udaljenost od nekog početnog čvora do svih ostalih.

U primjeru BFS prolaska kroz graf na slici 3.18, čvorovi 5 i 6 postavljeni su u rep prilikom obrade čvora 2. To znači da im je čvor 2 prethodnik na putu od početnog čvora do njih. Nadalje, neka je udaljenost čvora 2 u općem slučaju jednaka k. S obzirom da se do čvorova 5 i 6 dolazi preko čvora 2, i da je udaljenost od 2 do 5 i od 2 do 6 jednaka 1, onda je udaljenost čvorova 5 i 6 od početnog čvora jednaka $k+1$. U konkretnom primjeru $k+1=2$.

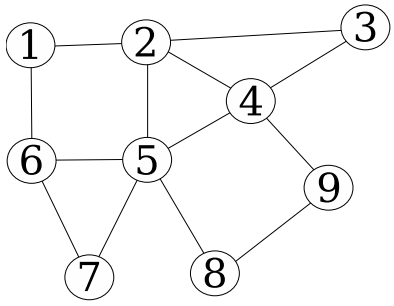
U dosadašnjim primjerima prikazani su prolasci kroz usmjereni graf. Za jednostavan graf se svojstva mijenjaju prilično očekivano tako da nema potrebe za detaljnijim obrazlaganjem. Napomenimo samo da za utvrđivanje povezanosti jednostavnog grafa nije potrebno izvoditi prolazak od svakog čvora ispočetka, već je dovoljno od bilo kojeg čvora proći po grafu. Ukoliko smo tim prolaskom obišli sve čvorove, to znači da je jednostavni graf povezan. Nadalje, s obzirom da podgraf generiran prolaskom kroz graf predstavlja stablo, postoji jedinstven put između svih parova čvorova.

Vezano uz problem stroge povezanosti grafa javlja se i sljedeći pojam. Ako u jednostavnom grafu G, svaku granu zamijenimo s lukom te ako je rezultirajući graf G' strogo povezan, tada graf G' nazivamo **stroga orijentacija** od G. Za jednostavan graf G kažemo da ga je moguće strogo orijentirati ako postoji stroga orijentacija tog grafa.

Teorem 3.11 (Robbins) Graf se može strogo orijentirati ako i samo ako je povezan i nema mostova.

Primjer 3.17 Gradska uprava nekog grada odlučila je sve ceste u središtu grada učiniti jednosmjernima u cilju povećanja protočnosti. Ceste su predstavljene jednostavnim grafom prikazanim na slici 3.19. Je li nakana gradske uprave ostvariva?

Rješenje: Prikazani graf je povezan i nema grana koje su most. Dakle, moguće ga je strogo orijentirati.



Slika 3.19: uz primjer 3.17, stroga orijentiranost

DFS algoritam se može jednostavno proširiti na način da se prilikom obrade čvorova bilježe „vremenski" trenuci u kojima su čvorovi po prvi puta posjećeni (dakle, kad su čvorovi stavljeni u stog) i trenuci u kojima su čvorovi u potpunosti obrađeni. Algoritam 3.7.2 prikazuje rekurzivnu varijantu DFS algoritma u kojem se koriste polja $d[v]$ i $f[v]$, gdje je $d[v]$ trenutak kad je čvor v otkriven, tj. kad je stavljen na stog, a $f[v]$ je trenutak kad je čvor v u potpunosti obrađen.

Jedna od primjena ovakvog algoritma bit će prikazana u sljedećem odjeljku.

3.8 Topološko sortiranje

DFS prolazak kroz graf nalazi čestu primjenu, što direktno što kao dio složenijih algoritama. Jedan primjer korištenja DFS algoritma je i u problemu topološkog sortiranja o kojem je bilo riječi u primjeru na slici 3.2. **Topološki uređaj (ili topološki sort)** usmjerenog acikličkog grafa G predstavlja slijed čvorova tog grafa takav da ako G sadrži granu (u,v) onda u sortiranom slijedu u prethodi v.

Usmjereni aciklički grafovi su česti u primjeni te se za njih ponekad koristi i skraćenica **dag** engleskog naziva engl. *directed acyclic graph*. Jedan algoritam za topološko sortiranje usmjerenih acikličkih grafova moguće je

algoritam 3.7.2 DFS prolazak, rekurzivna varijanta s vremenskim oznakama

Ulaz: $G = (V, E)$

Izlaz: podgraf prethodnika $G_\pi(V, E_\pi)$, gdje je $E_\pi = \{(\pi[v], v) : v \in V \text{ i } \pi[v] \neq 0\}$, $d[v]$ vrijeme početka obrade čvora $v \in G$, $f[v]$ vrijeme završetka obrade čvora $v \in G$

Lokalne varijable: S stog, $M[v] = \begin{cases} 0 & \text{netaknut} \\ 1 & \text{u stogu} \\ 2 & \text{obrađen} \end{cases}$

```
1: procedure DFS(G)
2: for svaki čvor u ∈ V do
3:    M[u] ← 0
4:    π[u] ← NULL
5: end for
6: time ← 0
7: for all u ∈ V(G) do
8:    if M[u] = 0 then
9:       call DFS_visit(G,u)
10:   end if
11: end for
12: end proc
13:
14: procedure DFS_visit(G,u)
15: M[u] ← 1
16: d[u] ← time ← time + 1
17: for all v susjedan čvoru u do
18:   if M[s] = 0 then
19:      π[s] ← w
20:      call DFS_visit(G, v)
21:   end if
22: end for
23: M[u] ← 2
24: f[u] ← time ← time + 1
```

ostvariti korištenjem, odnosno, proširenjem DFS algoritma. Algoritam 3.8.1 obavlja topološki sort korištenjem DFS algoritma.

Primjer 3.18 Provedimo prikazani algoritam na dagu sa slike 3.20.

Rješenje:

Topološko sortiranje može se izvesti i na jednostavniji način od algoritma 3.8.1. Ideja je sljedeća. Izaberimo u grafu bilo koji čvor koji ima ulazni stupanj 0, dodijelimo mu oznaku 1 i obrišimo sve grane koje iz njega izviru prema ostalim čvorovima. U rezultirajućem grafu, ponovo izaberimo proizvoljni čvor koji ima ulazni stupanj 0, dodijelimo mu oznaku 2 i obrišimo sve grane koje iz njega izviru. Ponavljajući ovaj postupak, ukoliko je usmjereni graf zaista acikličan, oznake dodijeljene čvorovima predstavljat će topološki uređaj.

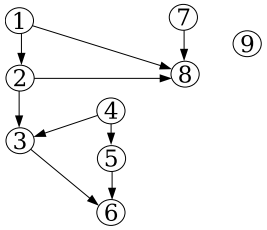
Algoritam 3.8.2 predstavlja realizaciju navedene ideje.

algoritam 3.8.1 Topološko sortiranje

Ulaz: usmjereni aciklički graf $G = (V, E)$

Izlaz: vezana lista L čvorova koja predstavlja topološki uređaj grafa G

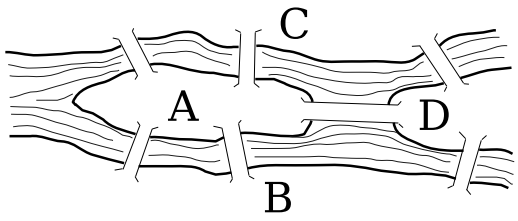
```
1: call DFS(G) da bi se izračunala vremena završataka obrade čvorova f[v]
2: u trenutku kad se v čvoru dodjeljuje vrijeme f[v], dodati čvor na početak vezane liste L
```



Slika 3.20: DAG

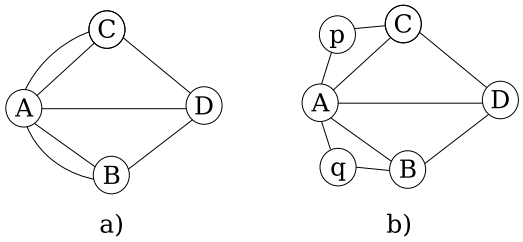
3.9 Eulerov ciklus

Prvi poznati problem u kojem je korišten graf kao matematički model datira iz 18. stoljeća i smatra se začetkom matematičke teorije grafova. U gradu Königsbergu, postojalo je sedam mostova koji su povezivali obale rijeke Pregel i dva otočića u sredini rijeke (slika 3.21). Mještane grada oduvijek je zabavljalo sljedeće pitanje: može li se, neprekinutim pješačenjem, prijeći svih sedam mostova, ali tako da se niti po jednom mostu ne prijeđe više od jedanput?



Slika 3.21: Mostovi Königsberga

Mostove i kopna koja oni povezuju možemo predstaviti multigrafom prikazanim na slici 3.22-a). (Na slici 3.22-b) prikazana je transformacija multigrafa u jednostavan graf koja ne mijenja njegova svojstva s obzirom na postojanje Eulerovih ciklusa.) U odjeljku 3.5 uveden je pojam *trag* koji predstavlja šetnju između dva čvora bez ponovljenih grana, tj. ne prolazeći po istoj grani više od jedanput. Dakle, postavljeno pitanje na modelu multigrafa bi glasilo: postoji li u multigrafu takav trag (može biti kolo, ali i ne mora) koji sadrži sve grane?



Slika 3.22: Multigraf Königsberških mostova

Iako su mještani Königsberga naslućivali da je odgovor na njihovo pitanje negativan, švicarski matematičar Euler je to i dokazao. Zbog toga, sljedeći niz pojmova nosi Eulerovo ime.

Ako trag između dvaju različitih čvorova u povezanom grafu (koji ne mora biti jednostavan) sadrži sve grane u grafu, nazivamo ga **Eulerov trag**. Kolo koje sadrži sve grane nazivamo **Eulerovo kolo**. Graf nazivamo **Eulerov graf** ako u njemu postoji Eulerovo kolo. Graf u kojem postoji Eulerov trag nazivamo polu-Eulerov graf.

Sljedeće teorem ukazuje na vrlo jednostavan način kojim je moguće odrediti da li je proizvoljan multigraf eulerovski, tj. da li posjeduje Eulerovo kolo.

Teorem 3.12 Multigraf G je eulerovski ako i samo ako je povezan te je svaki čvor u njemu paran (tj. stupanj

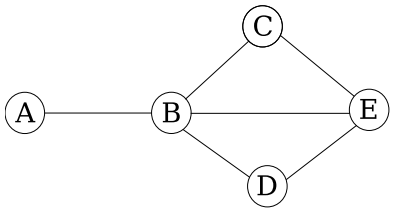
algoritam 3.8.2 Topološko sortiranje

Ulaz: usmjereni aciklički graf $G = (V, E)$
Izlaz: vezana lista L čvorova koja predstavlja topološki uređaj grafa G
lokalne varijable: `ulazni_stupanj[i]` ulazni stupanj čvora i

```
1:  procedure toposort(G)
2:  for ∀i ∈ V do
3:     ulazni_stupanj[i] ← 0
4:  end for
5:  for ∀(i, j) ∈ E do
6:     ulazni_stupan[j] ← ulazni_stupanj[j] + 1
7:  end for
8:  L ← ∅
9:  next ← 0
10: for ∀i ∈ V do
11:    if ulazni_stupanj[i] = 0 then
12:       L ← L ∪ {i}
13:    end if
14: end for
15: while L ≠ ∅ do
16:    uzeti prvi čvor iz L i obrisati ga
17:    next ← next + 1
18:    mjesto[i] ← next
19:    for ∀(i, j) ∈ A(i) do
20:       ulazni_stupanj[j] ← ulazni_stupanj[j] + 1
21:       if ulazni_stupanj[j] = 0 then
22:          L ← L ∪ {j}
23:       end if
24:    end for
25: end while
```

čvora je paran broj).

S obzirom da u multigrafu mostova K"onigsberga nisu svi čvorovi parni, zaključujemo da ne postoji takvo kolo koje prolazi po svim mostovima samo jednom i završava u početnoj točki. Međutim, mještani bi bili zadovoljni i s rješenjem koje ne završava u istoj točki, odnosno, dovoljno je bilo kako proći sve mostove bez prolaska po svakom više od jedanput. Dakle, zanimalo ih je postoji li Eulerov trag, ako već ne i Eulerovo kolo. Grafove u kojima postoji Eulerov trag (dakle, ne kolo) nazivamo **prohodnima**. Na primjer, graf na slici 3.23 predstavlja prohodan graf.



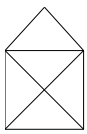
Slika 3.23: Prohodan graf

Sljedeći teorem rješava pitanje mostova Königsberga.

Teorem 3.13 Multigraf G je prohodan ako i samo ako je povezan i ako posjeduje *točno* dva neparna čvora.

Bitno je i sljedeće: svaki Eulerov trag u prohodnom grafu počinje u jednom od neparnih čvorova, a završava u drugom. S obzirom da svi čvorovi na slici 3.22 imaju neparan stupanj, taj multigraf nije niti Eulerov niti prohodan.

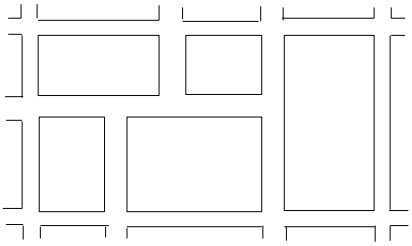
Primjer 3.19 Je li moguće nacrtati sliku 3.24 "u jednom potezu", tj. ne podižući olovku s papira, ali i ne crtajući linije više od jednom.



Slika 3.24:

Rješenje: Moguće je jer se slika može modelirati prohodnim grafom.

Primjer 3.20 Dostavljač pošte dostavlja poštu po dijelu grada u kojem su ulice raspoređene kao što je prikazano na slici 3.25. Ako dostavljač u jednom prolasku kroz ulicu poštu isporučuje samo na jednoj strani ulice, postoji li obilazak svih strana ulica u gradu takav da se po svakoj strani prolazi samo jednom.



Slika 3.25:

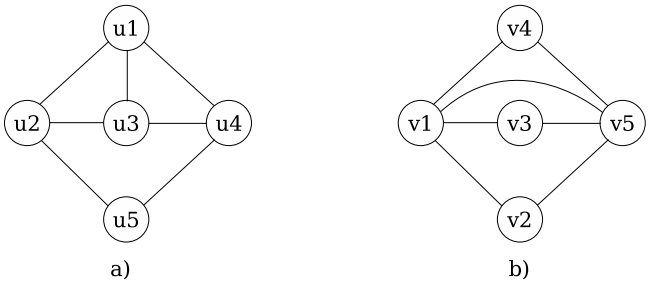
3.10 Hamiltonov ciklus

Sljedeći primjer predstavlja podlogu za definicije prikazane u ovom odjeljku.

Primjer 3.21 Neki trgovači putnik mora osobnim automobilom obići niz gradova u kojima nudi svoje proizvode. Postavlja se pitanje, postoji li takav obilazak gradova pri kojem će, počevši od proizvoljnog grada, kroz sve ostale gradove proći točno jednom i opet završiti u početnom gradu?

Ukoliko gradove i ceste predstavimo grafom G, pitanje glasi: postoji li ciklus u G koji sadržava sve čvorove. Graf G nazivamo **Hamiltonovim grafom** ako u njemu postoji ciklus koji sadrži sve čvorove u G. Takav ciklus nazivamo **Hamiltonovim ciklusom**. Hamiltonov ciklus je razapinjući ciklus. Ukoliko postoji put koji obuhvaća sve čvorove grafa, takav put nazivamo **Hamiltonov put** u grafu.

Na primjer, graf na slici 3.26-a) je Hamiltonov, dok graf na slici 3.26-b) to nije.



Slika 3.26: a) Hamiltonov graf, b) graf koji nije Hamiltonov

Da bismo dokazali da je graf a) Hamiltonov, dovoljno je naći jedan Hamiltonov ciklus. To je, na primjer, ciklus $u_1, u_2, u_5, u_4, u_3, u_1$. Međutim, nešto je teže dokazati da graf na slici b) nije Hamiltonov. Poslužit ćemo se

kontradikcijom. Pretpostavimo da graf jest Hamiltonov. Dakle, graf G sadrži Hamiltonov ciklus C. C sadrži svaki čvor u G pa tako i čvorove v_2, v_3 i v_4. Svaki od čvorova v_2, v_3 i v_4 ima stupanj 2. Dakle, ciklus C mora sadržavati obje grane incidente s v_2, v_3 i v_4. To, nadalje, znači da su u ciklusu C prisutne grane (v_1, v_2), (v_1, v_3) i (v_1, v_4). Međutim, svaki ciklus može sadržavati samo dvije grane incidente s nekim čvorom u tom ciklusu (granu kroz koju se „ulazi" u čvor i granu preko koje se „izašlo" iz čvora). Iz ovoga slijedi da graf ne može sadržavati Hamiltonov ciklus što je kontradiktorno s pretpostavkom da je graf Hamiltonov. Time je dokazano da graf na slici b) nije Hamiltonov.

Problem naveden u uvodu ovog poglavlja svodi se na utvrđivanje da li je pripadajući graf Hamiltonov ili ne.

Za razliku od Eulerovih grafova, ne postoji jednostavan način da bi se za proizvoljan graf utvrdilo da li je Hamiltonov ili ne. Određivanje tog svojstva potrebno je obaviti za svaki konkretni graf posebno. Međutim, postoji velik broj različitih dovoljnih uvjeta koji na pri tome mogu pomoći. Kao primjer navodimo dva takva uvjeta.

Teorem 3.14 (Ore) Jednostavan graf s $n > 2$ čvorova je Hamiltonov ako je zbroj stupnjeva svakog para nesusjednih čvorova veći ili jednak n.

Teorem 3.15 (Dirac) Jednostavan graf s $n > 2$ čvorova je Hamiltonov ako je stupanj svakog čvora veći ili jednak $n/2$.

3.11 Minimalno razapinjuće stablo

Ukoliko svakoj grani e u grafu G pridružimo realan broj $w(e)$ takav graf nazivamo **težinski graf**. Težinski graf označavamo s $G = (V, E, w)$ gdje je w preslikavanje $w : E \to \mathbb{R}$. Pridruženi realni brojevi nazivaju se **težine grana**. Za usmjerene grafove vrijede analogne definicije. Težinski graf ili digraf često se naziva **mreža**.

Ako je H podgraf težinskog grafa G, težina tog podgrafa definirana je kao zbroj težina svih grana u H i označava se s $w(H)$. Razapinjuće stablo T u težinskom grafu G nazivamo **minimalno razapinjuće stablo** grafa G ako je $w(T) \leq w(T')$ za svako razapinjuće stablo T' u G.

Problem određivanja minimalnog razapinjućeg stabla (engl. *minimum spanning tree, MST*) često se javlja u praksi.

Primjer 3.22 Zadan je skup gradova koje je potrebno povezati u jedinstvenu električnu mrežu izgradnjom novog sustava dalekovoda. Pretpostavimo da cijena izgradnje dionice dalekovoda ovisi samo o duljini dionice, tj. o udaljenosti između dva grada koje povezuje određena dionica dalekovoda. Potrebno je povezati gradove dalekovodima tako da je omogućena distribucija električne energije između svaka dva grada te da je ukupna cijena izgradnje minimalna.

Rješenje: Konačno rješenje predstavljat će minimalno razapinjuće stablo u potpunom grafu u kojem čvorovi odgovaraju gradovima, a težine grana jednake su udaljenostima među gradovima.

Dva algoritma koja rješavaju MST problem, a temelje se na *greedy* metodi su Kruskalov i Prim-Dijkstrin algoritam. U nastavku su prikazane idejne verzije ova dva algoritma.

3.11.1 Kruskalov algoritam

Održava šumu.

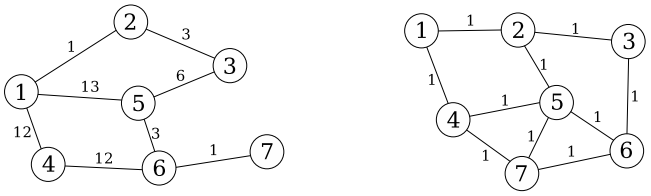
Nedovršeno **Nedovršeno**

3.11.2 Prim-Dijkstrin algoritam

Održava stablo.

Nedovršeno **Nedovršeno**

Primjer 3.23 Za grafove prikazane na slikama 3.27-a) i 3.27-b) odredite minimalna razapinjuća stabla. Kolike su težine stabala? Postoji li više rješenja?



Slika 3.27:

Primjer 3.24 Za graf prikazan na slici 3.27-a) pronađite maksimalno razapinjuće stablo, tj. razapinjuće stablo maksimalne težine.

3.12 Najkraći putovi

Potreba za pronalaženjem najkraćeg puta između dva čvora u grafu se pojavljuje vrlo često. Na primjer, u Internet mreži, usmjerivači moraju znati kojim putom usmjeravati pakete da bi oni što brže stigli na odredište. Nadalje, izbor najkraćeg puta do odredišnog grada je za vozača kamiona vrlo bitan.

U proizvoljnom težinskom grafu $G = (V, E, w)$, gdje je $w : E \to \mathbb{R}$, obično postoji više putova između svaka dva čvora. Težina (duljina) puta $p = \langle v_0, v_1, \ldots, v_k \rangle$ je zbroj težina (duljina) pojedinih grana na tom putu[3]:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Duljinu najkraćeg puta između čvorova u i v označavamo s $\delta(u, v)$ i je definirana na sljedeći način:

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

Težine grana mogu predstavljati različite fizikalne veličine kao što su vrijeme, gubitke, trošak ili bilo koju veličinu za koju ima smisla definirati da je težina puta jednaka zbroju težina grana u njemu, tj. za veličine koje se akumuliraju linearno duž puta.

BFS algoritam opisan u odjeljku 3.7 pronalazi najkraće puteve u ne-težinskom grafu. Ukoliko su duljine grana mali prirodni brojevi, jednostavnom transformacijom može se BFS koristiti i za pronalaženje puteva u ovakvim grafovima. Ukoliko svaku granu koja ima vrijednost duljine veću od 1 zamijenimo nizom grana i čvorova čiji broj odgovara duljini originalne grane, rješenje koje BFS objavi sadržavat će i rješenje početnog problema. (Probajte!) Međutim, ovaj algoritam ne može riješiti općeniti problem najkraćih puteva gdje su težine grana realni brojevi.

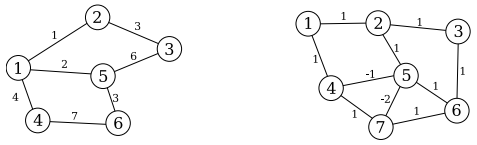
Postoji nekoliko varijanti problema najkraćih putova. Kao prvu varijantu možemo izdvojiti problem pronalaska najkraćeg puta između točno dva čvora. Druga varijanta bila bi pronalaženje najkraćih putova od jednog čvora do svih ostalih. Ova druga varijanta vrlo često se koristi u praksi. Na primjer, upravo to je problem s kojim se suočavaju usmjerivači u Internet mreži. Zanimljiva je činjenica da do danas nije poznat algoritam koji rješava prvu varijantu problema, a koji ima manju $O(\cdot)$ složenost najgoreg slučaja od algoritama koji rješavaju drugu varijantu problema. To praktično znači da je jednako teško pronaći najkraći put između točno dva čvora kao i pronaći sve najkraće putove od jednog čvora do svih ostalih. Treća varijanta problema pronalaženje najkraćih

[3]U težinskom grafu govorimo o težinama grana, pa stoga definiramo i težinu puta. Međutim, prirodnije je govoriti o duljini nekog puta, pa ćemo u nastavku teksta kao sinonime koristiti pojmove težina puta i duljina puta te pojmove težina grane i duljina grane.

putova između svih parova čvorova. Ova varijanta problema mogla bi se rješavati pomoću rješenja koja se koriste u drugoj varijanti na način da se pronađu najkraći putevi do svih ostalih čvorova za svaki od čvorova. Međutim, za treću varijantu postoje efikasniji algoritmi. U ovom odjeljku od interesa je rješenje druge varijante problema, tj. pronalaženje najkraćih putova od jednog čvora do svih ostalih.

U nekim primjenama mogu se pojaviti grane s negativnim težinama. Na primjer, ukoliko težine grana predstavljaju novčane troškove, negativna vrijednost grane može predstavljati dobitak. Ukoliko graf $G = (V, E)$ ne sadrži negativne cikluse (tj. ciklus čija težina je negativna) dostupne iz početnog čvora s, problem najkraćih puteva ne gubi smisao bez obzira što neke grane imaju negativne težine. Međutim, ukoliko postoji negativni ciklus dostupan iz s onda problem najkraćeg puta gubi smisao jer je za svaki put moguće naći još kraći. Stoga duljinu u tom slučaju definiramo s $\delta(s, v) = -\infty$.

Primjer 3.25 Graf na slici 3.28-a) ne posjeduje negativni ciklus, dok graf na slici 3.28-b) posjeduje negativni ciklus.



Slika 3.28: a) graf bez negativnog ciklusa, b) graf s negativnim ciklusom 4,5,7,4

U ovom odjeljku bit će obrađen Dijkstrin algoritam. Ovaj algoritam pretpostavlja da u mreži ne postoje grane s negativnim težinama. Za razliku od Dijkstrinog algoritma, Bellman-Fordov algoritam dozvoljava postojanje grana s negativnim težinama te algoritam ustanovljava i eventualno postojanje negativnih ciklusa.

3.12.1 Dijkstrin algoritam

Dijkstrin algoritam pronalazi najkraće putove od nekog početnog čvora do svih ostalih uz pretpostaku da su težine grana pozitivni realni brojevi. Ovaj algoritam temelji se na nekoliko svojstava samih najkraćih putova koja iznosimo u nastavku.

Teorem 3.16 Za zadani usmjereni težinski graf $G = (V, E, w)$, gdje je $w : E \to \mathbb{R}$, neka je $p = \langle v_0, v_1, \ldots, v_k \rangle$ najkraći put od čvora v_1 do v_k. Neka je, nadalje, za sve i i j takve da $1 \leq i \leq j \leq k$, $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ dio puta p od čvora i do j. Onda je p_{ij} najkraći put od v_i do v_j.

Dokaz. Ako rastavimo put p na tri dijela $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, onda je $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$. Pretpostavimo sada da postoji put p'_{ij} od v_i do v_j težine $w(p'_{ij}) < w(p_{ij})$. Onda je put $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ od v_1 do v_k čija duljina je $w(p) = w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$ kraći od $w(p)$ što je u kontradikciji s pretpostavkom da je p najkraći put od v_1 do v_k. □

Teorem 3.17 Neka je $G = (V, E, w)$ težinski graf. Pretpostavimo da se najkraći put p od izvornog čvora s do čvora v može rastaviti u $s \overset{p'}{\rightsquigarrow} u \to v$ za neki čvor u i put p'. Tada je težina najkraćeg puta od s do v jednaka $\delta(s, v) = \delta(s, u) + w(u, v)$.

Dokaz. Prema teoremu 3.16, podput p' je najkraći put od s do u. Dakle,

$$\begin{aligned} \delta(s, v) &= w(p) \\ &= w(p') + w(u, v) \\ &= \delta(s, u) + w(u, v) \end{aligned}$$

što dokazuje teorem. □

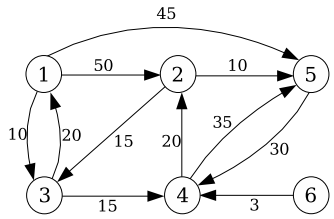
Osnovna ideja Dijkstrinog algoritma temelji se na *greedy* metodi i sastoji se u tome da se putovi pronalaze u neopadajućem redoslijedu njihovih duljina. To znači da se prvo pronalazi najkraći put do najbližeg čvora

od svih čvorova. Nakon toga se pronalazi najkraći put do sljedećeg najbližeg čvora i tako dalje. Prije same definicije algoritma, prikazat ćemo njegovu ideju na primjeru.

Neka je zadan usmjereni graf na slici 3.29. Redoslijed otkrivanja najkraćih putova je sljedeći:

Put	Duljina
1,3	10
1,3,4	25
1,3,4,2	45
1,5	45

To su najkraći putovi od čvora 1 do svih ostalih čvorova do kojih put postoji. Primijetite da do čvora 6 ne postoji put.



Slika 3.29:

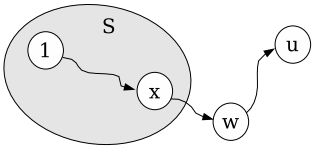
Da bismo generirali najkraće putove na opisani način, moramo u svakom trenutku moći odrediti:

1. sljedeći čvor do kojeg se mora generirati najkraći put
2. najkraći put do tog čvora

Označimo sa S skup čvorova (uključivo s početnim čvorom 1) do kojih smo već generirali najkraći put. Za čvor w koji nije u S, neka je $d[w]$ duljina najkraćeg puta koji počinje od čvora 1, prolazi samo kroz čvorove koji su u S i završava u w. Primjećujemo sljedeće:

1. Ako je sljedeći najkraći put put do čvora u, onda taj put počinje u čvoru 1, završava u čvoru u te prolazi samo kroz čvorove koji su u S.

   Da bismo to dokazali, moramo pokazati da svi međučvorovi u najkraćem putu moraju biti u S. Pretpostavimo da postoji čvor w na putu od 1 do u koji nije u S (vidi sliku 3.30). Tada, put od čvora 1 do u



Slika 3.30:

   također sadrži put od 1 do w koji je manje duljine od puta od 1 do u. Po pretpostavci, najkraći putovi se pronalaze u neopadajućem redoslijedu duljina što znači da je put od 1 do w već prije morao biti pronađen te ova kontradikcija dokazuje tvrdnju.

2. Čvor do kojeg se računa sljedeći najkraći put mora biti onaj čvor u koji ima najmanju $d[u]$ od svih čvorova koji nisu u S. Ovo slijedi iz definicije i prethodne točke.

3. Nakon što je izabran čvor u u skladu s prethodnom točkom, čvor u postaje dio skupa S. U ovom trenutku duljina najkraćeg puta od 1 preko čvorova iz S do nekog čvora w koji nije u S može se smanjiti, tj. vrijednost $d[w]$ se može smanjiti jer je S promijenjen. Ako se promijeni, onda to mora biti uslijed pronalaska kraćeg puta od 1 preko čvora u koji je upravo ušao u S.

   Međučvorovi na putu od 1 do u i na putu od u do w moraju svi biti u S. Put od 1 do u mora biti najkraći takav put. Također, put od u do w može biti izabran tako da ne sadrži međučvorove. Iz navedenog

zaključujemo: ako se $d[w]$ smanjuje onda je to zbog puta od 1 preko u do w gdje je put od 1 do u najkraći takav put, a put od u do w je grana (u, w). Duljina ovog puta je $d[w] = d[u] + w(u, w)$.

Algoritam 3.12.1 predstavlja formalizaciju navedenih ideja.

algoritam 3.12.1 Dijkstrin algoritam za najkraće putove u težinskom usmjerenom grafu

ulaz: graf $G = (V, E, w)$, $w : E \to \mathbb{R}^+$, početni čvor s
izlaz: niz $d[v]$ koji predstavlja duljinu najkraćeg puta od početnog čvora s do čvora v za svaki čvor v
pomoćne varijable: skup S čvorova za koje je utvrđen najkraći put, skup Q svih ostalih čvorova, tj. $Q = V \setminus S$, $A[v]$ lista susjeda čvora v

```
1: procedure Dijkstra(G)
2: {Inicijalizacija}
3: for ∀v ∈ V do
4:    d[v] ← ∞
5:    π[v] ← NULL
6: end for
7: d[s] ← 0 { Početni čvor }
8: S ← ∅
9: Q ← V
10: while Q ≠ ∅ do
11:    u ← min_{j∈T}{d[u_j]}
12:    S ← S ∪ {u}
13:    for ∀v ∈ A[u] do
14:       if d[v] > d[u] + w(u, v) then
15:          d[v] ← d[u] + w(u, v)
16:          π[v] ← u
17:       end if
18:    end for
19: end while
```

Složenost Dijkstrinog algoritma ovisi u implementaciji strukture podataka koja čuva skup T. Iz ovog skupa se mora dohvaćati najmanji element što nije jednostavna operacija. Uobičajeno je da se za implementaciju koristi binarno stablo te je složenost ovakve implementacije $O(|V| \log |V|)$.

Primjer 3.26 U telekomunikacijskoj mreži, vjerojatnost da grana $(i, j) \in E$ ispravno radi iznosi p_{ij}. Kako pronaći najpouzdaniji put između proizvoljna dva čvora takvoj mreži?

Rješenje: Vjerojatnost da sve grane u nekom putu ispravno rade jednaka je umnošku vjerojatnosti grana na tom putu. Ukoliko granama dodijelimo duljinu koja iznosi $a_{ij} = -\log p_{ij}$, problem se svodi na pronalazak najkraćeg puta između čvorova.

Primjer 3.27 Zamislite mehanički model mreže u kojoj su čvorovi predstavljeni kuglicama, a grane koncima koji povezuju kuglice. Također, duljina konca koji povezuje neka dva čvora neka odgovara duljini grane između ta dva čvora. Uz zadani početni čvor/kuglicu, kako iskoristiti opisani mehanički model da bi se pronašli najkraći putovi od početnog čvora do svih ostalih čvorova?

Rješenje: Potrebno je pustiti model da slobodno visi učvršćen u kuglici koja predstavlja početni čvor. Svi ostali čvorovi uzrokovat će napinjanje određenog broja konaca i to točno onih koji predstavljaju grane u nekom od najkraćih puteva od izvorišnog do svih odredišnih čvorova. Konci koji nisu u najkraćim putovima ostat će labavi.

3.12.2 Bellman-Fordov algoritam

Odnos među duljinama najkraćih puteva koji se prostiru od jednog (izvorišnog) čvora do svih ostalih uspostavljaju jednadžbe koje je postavio Bellman. Mreža je predstavljena usmjerenim grafom zadanim matricom

susjednosti čiji elementi imaju vrijednosti

$$w(u_i, u_j) = \begin{cases} \text{duljina grane } (u_i, u_j) & \text{ako grana postoji,} \\ +\infty & \text{inače} \end{cases}$$

Pretpostavimo da u grafu ne postoje ciklusi negativne duljine. Neka je izvorišni čvor označen oznakom u_1 i neka je za svaki drugi čvor $u_j, j \in \{2, 3, \ldots, n\}$

$$d[u_j] = \text{duljina najkraćeg puta od izvorišnog čvora do } u_j.$$

S obzirom da ne postoje zatvoreni ciklusi negativne duljine, sigurno je $d[u_1] = 0$. Prema teoremu 3.16, za najkraći put $p_n = \langle u_1, u_2, \ldots, u_k, \ldots, u_n \rangle$, svaki dio tog puta $p_k = \langle u_1, u_2, \ldots, u_k \rangle$ do nekog čvora u_k ujedno je i najkraći put do čvora u_k. To znači da za svaki od ostalih čvorova u_j za $j \neq 1$, najkraći put do tog čvora završava nekom granom (u_k, u_j), a s obzirom da je $d[u_k]$ duljina najkraćeg puta do čvora u_k, vrijedi da je $d[u_j] = d[u_k] + w(u_k, u_j)$. Također, u_k mora biti onaj čvor za koji je $d[u_k] + w(u_k, u_j)$ najmanje od svih čvorova. Zaključak je da za duljine najkraćih puteva moraju vrijediti sljedeće jednadžbe:

$$\begin{aligned} d[u_1] &= 0 \\ d[u_j] &= \min_{u_k \neq u_j} \{d[u_k] + w(u_k, u_j)\}, \qquad \forall j \in \{2, 3, \ldots, n\} \end{aligned} \tag{3.1}$$

Ovaj sustav od n jednadžbi naziva se Bellmanovim jednadžbama. Rješenje ovog sustava je konačno i jedinstveno. Jedan od algoritama koji se direktno temelje na ovim jednadžbama je i Bellman-Fordov algoritam prikazan u bloku3.12.2.

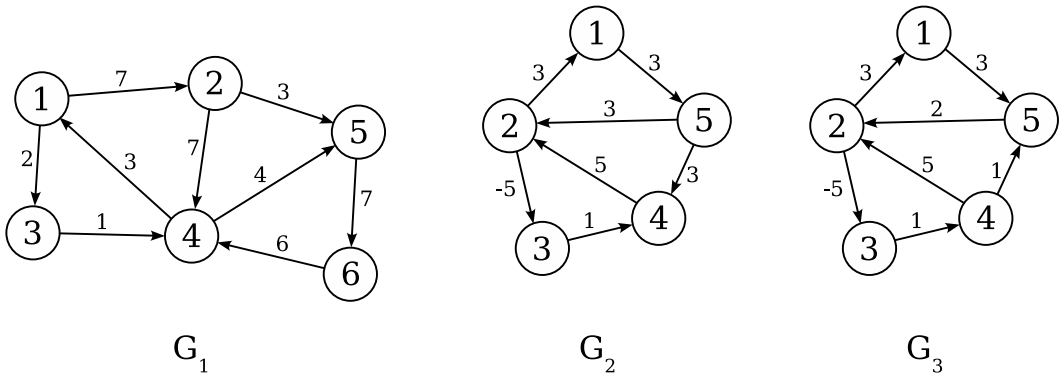
Složenost Bellman-Fordovog algoritma iznosi $O(VE)$ jer inicijalizacija (linije 2–6) uzima $O(V)$ vremena, svaki od $|V| - 1$ koraka `for` petlje uzima $O(E)$ vremena (linije 7–14), a zadnja `for` petlja uzima $O(E)$ vremena.

algoritam 3.12.2 Bellman-Fordov algoritam za najkraće puteve

ulaz: Graf $G = (V, E)$ s $|V|$ čvorova i $|E|$ grana, $w : E \to \mathbb{R}$, izvorišni čvor obilježen oznakom s
izlaz: Vrijednost `true` ukoliko graf ne sadrži negativne cikluse, udaljenosti $d[v]$ te graf prethodnika $\pi[v]$, $\forall v \in V$, od izvorišnog čvora s. Vrijednost `false` ukoliko graf sadrži negativni ciklus.

```
1:  procedure BellmanFord( G, s )
2:  for ∀v ∈ V do
3:      d[v] ← ∞
4:      π[v] ← NULL
5:  end for
6:  d[s] ← 0 { Početni čvor }
7:  for i ← 1 ... (|V| − 1) do
8:      for ∀(u, v) ∈ E do
9:          if d[v] > d[u] + w(u, v) then
10:             d[v] ← d[u] + w(u, v)
11:             π[v] ← u
12:         end if
13:     end for
14: end for
15: for ∀(u, v) ∈ E do
16:     if d[v] > d[u] + w(u, v) then
17:         return false
18:     end if
19: end for
20: return true
```

Primjer 3.28 Za grafove na slici 3.31, koristeći Bellman-Fordov algoritam pronađite najkraće putove od čvora 1 do svih ostalih čvorova.



Slika 3.31:

3.13 Razni primjeri

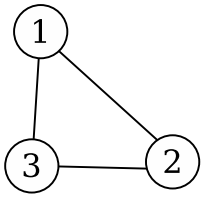
Primjer 3.29 Koliki je najveći mogući broj grana u jednostavnom grafu reda n?

Rješenje: $n(n-1)/2$. Svaki od n čvorova može imati po jednu granu prema svakom od ostalih $(n-1)$ čvorova. U jednostavnom grafu, grana (a,b) je isto što i grana (b,a) te izraz $n(n-1)$ predstavlja dvostruko više grana jer je svaka grana prebrojena dvaput: jednom iz čvora a prema b, a drugi put od b do a.

Primjer 3.30 Navedite dva grafa reda 5 koji međusobno nisu izomorfni.

Primjer 3.31 Navedite dva grafa reda 5 koji su međusobno izomorfni te imaju isti skup čvorova.

Primjer 3.32 Navedite barem jedan jednostavan graf koji je s grafom na slici 3.32 izomorfan, a nije mu identičan.



Slika 3.32:

Rješenje: Potpuni graf kod kojeg je skup čvorova $V = \{a, b, c\}$.

Primjer 3.33 Dokažite da jednostavni graf kod kojeg svaki čvor ima barem dvije grane ne mora biti povezan.

Rješenje: Sljedeći protuprimjer dokazuje tvrdnju: graf koji se sastoji od dvije potpuno povezane komponente reda 3. $G = (V, E)$, $V = \{1, 2, 3, 4, 5, 6\}$, $E = \{(1,2), (2,3), (3,1), (4,5), (5,6), (6,4)\}$.

Primjer 3.34 Navedite primjer samo-komplementarnog grafa reda 4.

Rješenje: $G = (V, E)$ gdje je $V = \{1, 2, 3, 4\}$, a $E = \{(1,2), (2,3), (3,4)\}$.

Primjer 3.35 Za graf prikazan na slici 3.34 navedite dva njegova podgrafa reda 2 koji međusobno nisu izomorfni.

Rješenje: $G_1 = (\,\{1,2\},\, \{(1,2)\}\,)$ i $G_2 = (\,\{1,2\},\, \emptyset\,)$.

Primjer 3.36 Koliko ima međusobno neizomorfnih cikličkih grafova reda 6?

Rješenje: Svaka dva ciklička grafa reda $n > 2$ su međusobno izomorfni.

Primjer 3.37 Koliki je minimalni broj grana potreban u telekomunikacijskoj mreži s $n > 2$ čvorova da bi se osiguralo da između svaka dva čvora postoje barem dva nezavisna puta.

Rješenje: Minimalni broj grana iznosi n. Graf mora biti povezan. U jednostavnom grafu reda $n > 2$, najmanji broj grana da bi graf uopće bio povezan iznosi $n - 1$ i to je onda stablo (prema teoremu 3.9). U stablu postoji jedinstven put između svaka dva čvora (također prema teoremu 3.9). Dakle, broj grana mora biti strogo veći od $n - 1$. Potrebno je još dokazati da je za svaki n moguće konstruirati graf reda n i veličine n koji između svaka dva čvora ima barem dva nezavisna puta. Rješenje je ciklički graf reda n. U telekomunikacijama se takve mreža nazivaju prstenastne mreže.

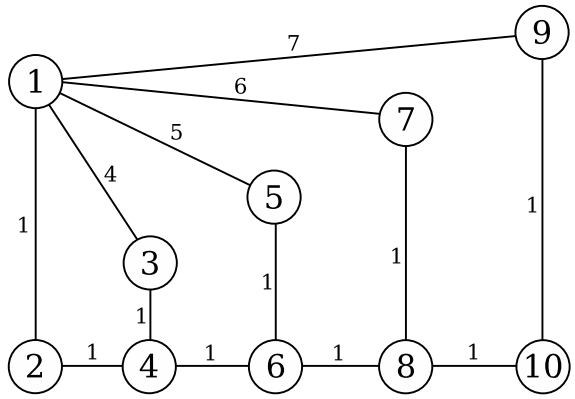
Primjer 3.38 Za graf K_6 odredite obilazak čvorova koristeći DFS i BFS algoritme počevši od proizvoljnog čvora. Postoji li u takvom grafu redoslijed obilaska korištenjem DFS algoritma koji nije moguće ostvariti korištenjem BFS algoritma?

Rješenje: Ne. Već u prvom koraku svi čvorovi odlaze na stog ili u rep. Redoslijed kojim odlaze u odgovarajuću strukturu je proizvoljan. Nakon toga prvog puta, više novi čvorovi neće biti otkriveni tako da redoslijed ovisi samo o prvom koraku.

Primjer 3.39 Na nekom geografskom području izgrađen je niz postaja koje imaju potrebu za međusobnom razmjenom podataka pri čemu koriste bežične, usmjerene odašiljače. Svaka od postaja ima sposobnost prosljeđivanja podataka, tj. podaci koji su do nje stigli, a namijenjeni su nekoj drugoj postaji, u stanju je ispravno proslijediti sljedećoj postaji na najkraćem putu do stvarnog odredišta. Za vezu između svake dvije postaje iskoristi se po jedan odašiljač u svakoj od postaja. Pretpostavimo da utrošak električne energije potrebne za održavanje veze između svaka dva odašiljača ovisi isključivo o njihovoj međusobnoj geografskoj udaljenosti. Isplanirajte veze među postajama tako da ukupna energija koja se na njih troši bude minimalna.

Rješenje: Minimalno razapinjuće stablo.

Primjer 3.40 Pronađite najkraće putove od čvora 1 do svih ostalih čvorova na slici 3.33 koristeći Dijkstrin algoritam.

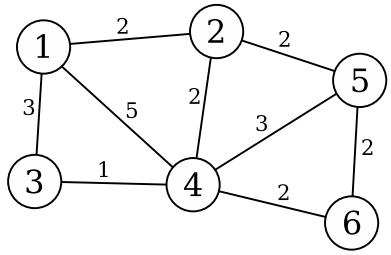


Slika 3.33:

Primjer 3.41 OSPF protokol. Dijkstrin algoritam koristi se u *Open Shortest Path First* protokolu za usmjeravanje u Internet mreži. Slijedi pojednostavljeni opis protokola. Svaki od čvorova u mreži svim ostalim čvorovima dostavlja informaciju o granama do kojih ima direktan pristup. Kad neki čvor od svih ostalih dobije informaciju o postojećim granama, on formira graf te nad tim grafom izvodi Dijkstrin algoritam proračunavajući najkraće putove od sebe do svih ostalih čvorova u mreži. Težina pojedine grane postavlja se od strane administratora mreže i obično je obrnuto proporcionalna kapacitetu grane. Kad čvor (IP usmjerivač, engl. *router*) dobije IP paket koji nosi odredišnu adresu nekog od ostalih čvorova, na temelju Dijkstrinog proračuna utvrđuje koji je njemu susjedan čvor ujedno sljedeći čvor na najkraćem putu do tog odredišta te se paket prosljeđuje tom susjedu. Susjed se ponaša na isti način te paket nakon konačnog broja skokova stiže na odredište.

Primjer 3.42 RIP protokol. Distribuirana verzija Bellman-Fordovog algoritma koristi se u *Routing Information Protocol* protokolu za usmjeravanje u Internet mreži. Slijedi pojednostavljeni opis protokola. Svaki od čvorova u mreži sa svojim susjedima razmjenjuje tzv. vektor udaljenosti. Vektor udaljenosti u nekom čvoru s je mapa u kojoj je svakom drugom čvoru u mreži dodijeljen broj koji predstavlja udaljenost od čvora s. Susjedni čvorovi međusobno razmjenjuju vektore udaljenosti i na taj način ustanovljavaju sve bolje i bolje putove do ostalih čvorova u mreži. Na primjer, pretpostavimo da neki čvor s_1 do nekog drugog nesusjednog čvora d trenutno zna za put duljine 8, odnosno u njegovom vektoru udaljenosti za čvor d pohranjena je vrijednost 8. Nadalje, neka čvor s_2 koji je s čvorom s_1 susjedan, čvoru s_1 isporuči svoj vektor udaljenosti u kojem tvrdi da on do odredišta ima 4. Ukoliko je duljina grane (s_1, s_2) jednaka, na primjer, 2, to za čvor s_1 znači da je put do d preko čvora s_2 dug 6. U skladu s tom novom infomacijom, čvor s_1 počinje koristiti put preko s_2 te svim svojim susjedima objavljuje ažurirani vektor udaljenosti u kojem je čvoru d pridružena vrijednost 6, a ne više 8.

Primjer 3.43 U mreži prikazanoj na slici 3.34 od čvora 1 do čvora 6 pronađite primarni (najkraći) put i jedan zaštitni koji je s primarnim disjunktan, tj. skup grana primarnog i skup grana zaštitnog puta su disjunktni skupovi. Formalizirajte korišteni algoritam.



Slika 3.34:

Primjer 3.44 U mreži prikazanoj na slici 3.34 pronađite prva dva najkraća (ne nužno disjunktna) puta od čvora 1 do čvora 6. Pokušajte formalizirati korišteni algoritam.

Primjer 3.45 OSPF protokol koristi statičke težine grana, tj. jednom dodijeljeni iznos težine se za vrijeme rada protokola ne mijenja. Razmislite što bi se događalo s prometom u mreži kad bi se težina grana dinamički mijenjala s obzirom na njihovo opterećenje.

Rješenje: Vrlo vjerojatno bi se pojavile oscilacije čiji period bi bio ovisan o periodu rasprostiranja informacije o opterećenosti grana među čvorovima. Kad je jedna od grana izrazito podopterećena, proračuni najkraćih putova od strane velikog broja čvorova počeli bi koristiti tu granu. To dovodi do zagušenja grane te ju svi čvorovi počinju izbjegavati. Ovo, pak, dovodi do podopterećenja grane, koju nakon nekog vremena opet svi počnu koristiti, što predstavlja oscilaciju prometa na toj grani.

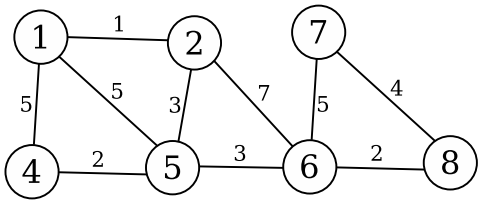
Primjer 3.46 Zadana je TK mreža na slici 3.35. Težine grana predstavljaju raspoloživ kapacitet grane (u bit/s). Osmislite algoritam koji će, počevši od proizvoljnog čvora, pronaći put s najmanje skokova po kojem je moguće uspostaviti rezervaciju od B bit/s kapaciteta.

Rješenje: Prvo se iz mreže izbace sve grane koje imaju kapacitet manji od B. Nakon toga se izvrti BFS algoritam i pronađu najkraći putevi do svih ostalih čvorova. Svaki od tih puteva ima kapacitet barem B.

Primjer 3.47 Kako bi se promijenio algoritam iz primjera 3.46 ukoliko bi se tražili najkraći putovi s obzirom na vrijeme propagacije, a svaka grana imala pridružen kapacitet i vrijeme propagacije?

3.14 Tokovi u mreži

Kod problema opisanih u prethodnim odjeljcima, težina grana u grafu predstavljale su duljine grana te su se tražili najkraći putovi između pojednih čvorova. Međutim, težina grane može predstavljati kapacitet grane za



Slika 3.35:

prijenos neke materije ili podataka po njoj. Na primjer, u telekomunikacijskoj mreži, svaka grana između dva čvora ima kapacitet koji predstavlja količinu podataka koja može po toj grani proći u jedinici vremena. Sâm prolazak podataka po granama mreže možemo zamisliti kao tok podataka u pojedinim granama. Uz ovakvu interpretaciju grafa, nameće se sljedeće pitanje. Koliki je najveći tok koji je moguće uspostaviti između dva čvora u mreži? Prije nego što krenemo u rješavanje ovog problema, definirat ćemo neke pojmove i pomoću njih formalizirati problem.

Mreža tokova $G = (V, E)$ je usmjereni graf u kojem svaka grana $(u, v) \in E$ ima dodijeljen ne-negativni **kapacitet** $c(u, v) \geq 0$. Ako je $(u, v) \notin E$, pretpostavit ćemo da je $c(u, v) = 0$. U mreži tokova izdvojena su dva čvora: **izvorišni** čvor s i **odredišni** čvor t. Graf je povezan te je $|E| \geq |V| - 1$.

Slika 3.36 prikazuje primjer mreže tokova.

Sada možemo formalno definirati tok u mreži. Neka je $G = (V, E)$ mreža tokova. Neka su s i t izvorišni i odredišni čvorovi u G. **Tok** u G je realna funkcija $f : V \times V \to \mathbb{R}$ koja posjeduje sljedeća tri svojstva:

Uvjet kapacitivnosti: $f(u, v) \leq c(u, v), \quad \forall u, v \in V$

Antisimetričnost: $f(u, v) = -f(v, u), \quad u, v \in V$

Očuvanje toka: Za sve $v \in V \setminus \{s, t\}$ vrijedi

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

Veličina $f(u, v)$ koja može biti pozitivna ili negativna naziva se **neto tok** od u do v. **Iznos toka** f se definira s

$$|f| = \sum_{v \in V} f(s, v)$$

i zapravo predstavlja neto tok koji izvire iz čvora s. Oznaka $|\cdot|$ predstavlja iznos toka, a ne apsolutnu vrijednost ili kardinalni broj. U problemu najvećeg toka navedenog u uvodnom dijelu, traži se tok između s i t s maksimalnim iznosom.

Uvjet kapacitivnosti znači da tok od jednog do drugog čvora ne može biti veći od kapaciteta grane koja ih povezuje. Primijetite da ako ne postoji grana između dva čvora, onda je po pretpostavci kapacitet 0 te tok ne smije biti veći od 0.

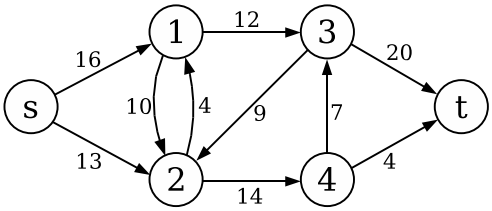
Antisimetričnost znači da je neto tok od nekog čvora u do nekog čvora v jednak negativnom neto toku od v do u. Iz ovoga slijedi da je neto tok od čvora prema samom sebi jednak 0 jer je $\forall v \in V$, $f(u, u) = -f(u, u)$ samo ako je $f(u, u) = 0$.

Uvjet očuvanja toka kaže da je iznos neto toka za svaki čvor v koji nije izvorišni ili odredišni jednak 0.

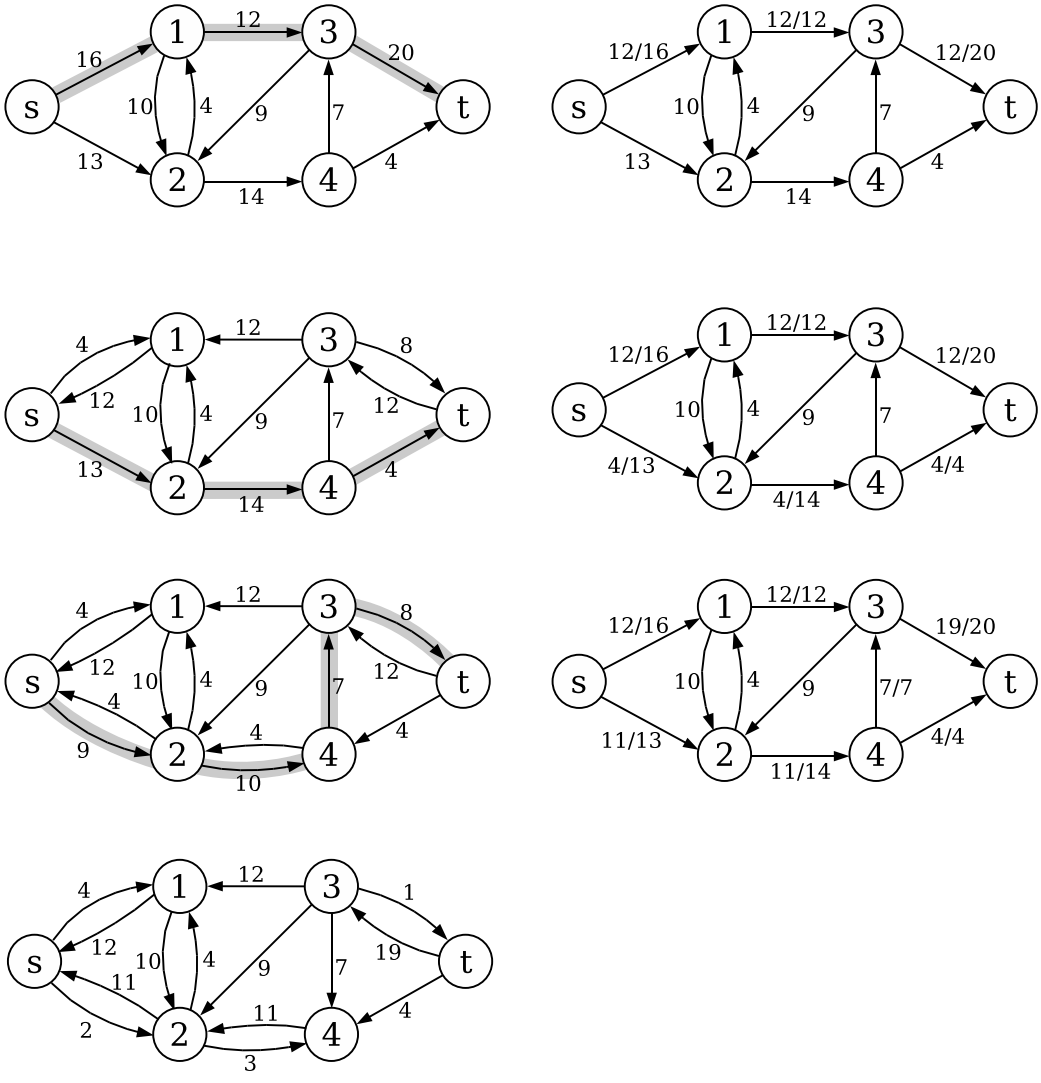
Uvedimo još jedan bitan pojam. **Pozitivni neto tok** koji ulazi u čvor v definira se s

$$\sum_{u \in V,\, f(u,v) > 0} f(u, v).$$

Primjer 3.48 Neki proizvođač sladoleda proizvodi sladoled u mjestu s, a prodaje ga u trgovini koja se nalazi u mjestu t. Proizvođač za prijevoz sladoleda od s do t koristi prijevozničku tvrtku koja između određenog skupa mjesta može ponuditi kapacitete koji su prikazani na slici 3.36.



Slika 3.36: Raspoloživi kapaciteti za prijevoz sladoleda u primjeru 3.48



Slika 3.37:

Poglavlje 4

Linearno programiranje

Optimizacijski problem je problem u kojem u određenom skup mogućih rješenja problema svako rješenje ima pridružen realan broj nazvan vrijednost te se traži ono rješenje koje ima najmanju vrijednost. Dakle, uz S kao skup mogućih rješenja te funkciju $f : S \to \mathbb{R}$, traži se minimalni element skupa $f(S)$.

Na primjer, neka je $S \subseteq \mathbb{R}^2$ skup rješenja koje zadovoljavaju sljedeće uvjete

$$\begin{aligned} x_1 - x_2 &= 3 \\ x_2 &\geq 2 \end{aligned}$$

gdje su $x_1, x_2 \in \mathbb{R}$. Dakle, S se sastoji od svih parova realnih brojeva x_1 i x_2 za koje vrijede navedeni uvjeti.

Jedan primjer optimizacijskog problema glasio bi: uz navedene uvjete, pronaći par brojeva za koje funkcija

$$f(x_1, x_2) = x_1^2 + x_2^2$$

poprima minimalnu vrijednost.

4.1 Matematički program

Matematički program je optimizacijski problem u kojem su funkcija cilja i uvjeti zadani u obliku matematičkih funkcija i odnosa među funkcijama. Formalno, matematički programi imaju sljedeći oblik:

Optimizirati (tj. pronaći minimum funkcije)

$$z = f(x_1, x_2, \ldots, x_n)$$

uz uvjete

$$g_i(x_1, x_2, \ldots, x_n) \quad \leq, =, \geq \quad b_i \quad \text{za} \quad i = 1, \ldots, m \tag{4.1}$$

Funkciju za koju tražimo minimalnu vrijednost nazivamo **funkcija cilja**.

Optimizacijski problem može biti definiran i na način da se umjesto minimalne vrijednosti traži maksimalna vrijednost funkcije cilja. S obzirom na to, razlikujemo minimizacijske i maksimizacijske probleme. Ova dva podtipa optimizacijskih problema su u osnovi ekvivalentni jer se maksimizacijski problem lako prevodi u minimizacijski i obrnuto. Na primjer, problem pronalaska maksimuma funkcije $f(\vec{x})$ ekvivalentan je pronalasku minimuma funkcije $-f(\vec{x})$.

S obzirom na oblik funkcije cilja i uvjeta, razlikujemo više vrsta matematičkih programa. Neki od njih prikazani su u sljedećim odjeljcima.

4.1.1 Linearni program

Matematički program nazivamo linearnim ako su i $f(x_1, \ldots, x_n)$ i svaki $g_i(x_1, x_2, \ldots, x_n)$ linearne funkcije u svojim argumentima, tj. ako se daju napisati u obliku

$$f(x_1, x_2, \ldots, x_n) = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

i

$$g_i(x_1, x_2, \ldots, x_n) = a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n$$

gdje su c_j i a_{ij} poznate konstante.

Svaki drugi matematički program je *nelinearni* program.

4.1.2 Cjelobrojni linearni programi

Cjelobrojni linearni program je linearni program s dodatnim ograničenjem da su sve varijable cijeli brojevi, dakle, iz skupa $\mathbb{Z}$. Ovakvo ograničenje se ne da izraziti funkcijski, nego se mora izreći riječima.

4.1.3 Kvadratični programi

Uvjeti linearni, funkcija cilja oblika:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_i x_j + \sum_{i=1}^{n} d_i x_i$$

4.2 Postupak formulacije linearnih programa

Optimizacijski problemi su najčešće zadani riječima. Postupak rješavanja se sastoji u modeliranju problema matematičkim programom te rješavanju matematičkog programa nekim od postojećih algoritama[1].

Postupak formulacije matematičkog programa može biti sljedeći:

1. Odrediti veličinu koja se optimizira i izraziti ju u obliku matematičke funkcije. Na taj način i definiramo ulazne varijable.
2. Identificirati sve uvjete i ograničenja i izraziti ih matematički.
3. Identificirati skrivene odnosno implicitne uvjete. To mogu biti uvjeti nenegativnosti, cjelobrojnositi i sl.

Primjer 4.1 Mesar proizvodi pljeskavice miješajući teletinu i svinjetinu. Teletina sadrži 80% mesa i 20% masnoće, a cijena joj je 80kn/kg. Svinjetina sadrži 68% mesa i 32% masnoće, a cijena joj je 60kn/kg. Koje i koliko mesa koristiti u svakoj kili pljeskavica da bi cijena pljeskavice bila što manja, a da udio masnoće u njoj ne prelazi 25%.

Rješenje: Odrediti cilj i izraziti ga funkcijom. Cilj je, dakle, minimizirati cijenu jedne kile pljeskavica izraženu u kunama.

$$z = 80 \cdot \text{masa upotrebljene teletine} + 60 \cdot \text{masa upotrebljene svinjetine}$$

Ako s x_1 označimo količinu upotrebljene teletine u kili pljeskavice (tj. kili smjese za pljeskavice), a s x_2 označimo količinu svinjetine, funkcija cilja se može izraziti s

$$z = 80x_1 + 60x_2$$

Svaka kila pljeskavice će sadržavati $0.20x_1$ kila masnoće koje donosi teletina te $0.32x_2$ kila masnoće koje donosi svinjetina. Ukupna količina masnoće u kili pljeskavice ne smije prelaziti 0.25%, tj. 0.25kg jer se sve računa za jednu kilu pljeskavice. Dakle,

$$0.20x_1 + 0.32x_2 \leq 0.25$$

[1] Pri rješavanju postavljenih matematičkih programa mogu nam pomoći računalni alati kao što su AMPL[?], LINDO[?] i mnogi drugi.

Masa upotrebljene teletine i svinjetine mora biti jedna kila, dakle,

$$x_1 + x_2 = 1$$

Implicitni (skriveni) uvjeti su: količina upotrebljenog mesa ne smije biti negativna. Dakle,

$$x_1 \geq 0, \qquad x_2 \geq 0$$

Znači, kompletan linearni program glasi:

$$\text{minimizirati} \quad z = 80x_1 + 60x_2$$

uz uvjete:

$$0.20x_1 + 0.32x_2 \leq 0.25$$
$$x_1 + x_2 = 1$$
$$x_1 \geq 0, \qquad x_2 \geq 0$$

Riješimo taj linearni program grafički!

Schaum 1.2.

Nedovršeno **Nedovršeno**

Primjer 4.2 Stolar ima na raspolaganju 6 komada drveta i 28 radnih sati za izradu drvenih figurica. Dva modela figurica su se u prošlosti dobro prodavala pa će se ograničiti na te modele. Procijenio je da je za izradu jedne figurice modela I potrebno 2 komada drveta i 7 sati rada, dok je za izradu modela II potrebno 1 komad drveta i 8 sati rada. Cijene po kojima se modeli prodaju su 120kn za model I i 80kn za model II i zna se da će svaka proizvedena figura biti prodana. Koliko figurica i od koje vrste napraviti da bi se maksimizirala dobit od prodaje.

Rješenje: Cilj je maksimizirati zaradu prodajom figurica. Zarađeni novac može se izraziti na sljedeći način:

$$z = 120 \cdot \text{broj proizvedenih figurica modela I} + 80 \cdot \text{broj proizvedenih figurica modela II}$$

Neka su

$$x_1 = \text{broj proizvedenih figurica modela I}$$
$$x_2 = \text{broj proizvedenih figurica modela II}$$

Cilj je:

$$\text{maksimizirati} \quad z = 120x_1 + 80x_2$$

Prvo ograničenje odnosi se na broj komada drveta koji stoje na raspolaganju.

$$2x_1 + x_2 \leq 6$$

Drugo ograničenje odnosi se na broj radnih sati.

$$7x_1 + 8x_2 \leq 28$$

Skriveni uvjeti su sljedeći. Očito je da x_1 i x_2 ne smiju biti negativni jer to nema smisla. Nadalje, ne mogu se prodavati djelomično završene figurice, odnosno, x_1 i x_2 moraju biti cijeli brojevi u konačnom rješenju.

Dakle, konačan cjelobrojni program glasi:

$$\text{maksimizirati} \quad z = 120x_1 + 80x_2$$

$$2x_1 + x_2 \leq 6$$
$$7x_1 + 8x_2 \leq 28$$
$$x_1 \geq 0, \quad x_2 \geq 0$$
$$x_1, x_2 \in \mathbb{N}_0$$

Rješimo problem grafički! Schaum 1.4.

Primjer 4.3 Planinar se sprema u brda. Postoji pet stvari koje bi htio ponijeti sa sobom, ali svih pet stvari teži više od 60kg koje je sposoban nositi za vrijeme puta. Da bi si olakšao izbor, svakoj stvari dodijelio je vrijednost koju ta stvar za njega ima.

Stvar	1	2	3	4	5
Masa	52	23	35	15	7
Vrijednost	100	60	70	15	15

Pitanje glasi: koje stvari treba ponijeti na put da bi se ponijela maksimalna vrijednost, a da se ne prijeđe težina od 60kg.

Rješenje: Neka x_i označava količinu stvari i koju se nosi na put. Iako je očito da x_i može poprimiti samo vrijednosti 0 ili 1, tj. što označava da li pojedinu stvar koju ostavljamo ili ju uzimamo na put, taj uvjet ostavljamo za kraj i radimo s realnim x_i. Tada je funkcija cilja koju je potrebno maksimizirati

$$\max z = 100x_1 + 60x_2 + 70x_3 + 15x_4 + 15x_5$$

Težina ponesenih stvari ne smije biti veća od 60kg, tj.

$$52x_1 + 23x_2 + 35x_3 + 15x_4 + 7x_5 \leq 60.$$
$$x_1 \leq 1$$
$$x_2 \leq 1$$
$$x_3 \leq 1$$
$$x_4 \leq 1$$
$$x_5 \leq 1$$
$$x_i \geq 0, \qquad x_i \in \mathbb{N}_0.$$

Primjer 4.4 Trgovački centar koji radi 24-sata dnevno ima sljedeće zahtjeve za blagajnicima:

Period	1	2	3	4	5	6
Vrijeme u danu	3-7	7-11	11-15	15-19	19-23	23-3
Min.broj blagajnika	7	20	14	20	10	5

Svaki blagajnik radi 8 sati u komadu, a počinje s radom na početku jednog od perioda. Utvrdite minimalni raspored zaposlenih po periodima koji zadovoljava zahtjev za minimalnim brojem blagajnika.

Rješenje: Neka je $x_i \quad (i = 1, 2, \ldots, 6)$ broj blagajnika koji započinju rad na početku perioda i. Matematički program glasi:

$$\min z = x_1 + x_2 + x_3 + x_4 + x_5 + x_6$$

uz uvjete

$$x_1 + x_6 \geq 7$$
$$x_1 + x_2 \geq 20$$
$$x_2 + x_3 \geq 14$$

$$x_3 + x_4 \geq 20$$
$$x_4 + x_5 \geq 10$$
$$x_5 + x_6 \geq 5$$

$$x_i \geq 0, \qquad x_i \in \mathbb{N}_0.$$

Primjer 4.5 Proizvođač igračaka proizvodi dječje lutke u gradovima D, C i I te ih prodaje u gradovima N, A, L. Trošak prijevoza između gradova je dan u tablici.

			Prema	
		N	A	L
	D	2.20	2.10	2.40
Od	C	1.80	1.90	2.10
	I	3.00	3.20	3.60

Proizvodni kapaciteti su u gradovima D, C i I redom: 250, 300, 200. Potražnja na prodajnim mjestima N, A i L iznosi redom: 190, 240 i 320. Kako distribuirati lutke da bi se ostvario minimalni trošak?

Rješenje:

Nedovršeno **Nedovršeno**

Primjer 4.6 Potrebno je obaviti četiri posla. Svaki od poslova može biti dodijeljen nekom od četvero radnika. Troškovi koje pri izradi posla potrebuju radnici prikazani su u tablici.

		Posao			
		1	2	3	4
	1	2	3	4	3
	2	5	4	3	2
Radnik	3	5	4	3	6
	4	7	3	2	2

Pronađite raspored radnika po poslovima koji osigurava minimalan trošak.

Rješenje:

Nedovršeno **Nedovršeno**

Primjer 4.7 Zadana je mreža na slici **??**. Svakoj grani pridružen je kapacitet. Postavite problem pronalaska maksimalnog mogućeg toka od čvora 0 do čvora m kao linearni program.

Rješenje:

4.3 Standardni oblik linearnog programa

Metode za rješavanje linearnih programa kao preduvjet zahtijevaju da se linearni program napiše u tzv. *standardnom obliku*. Standardni oblik formira se tako da se sve nejednakosti u uvjetima transformiraju u jednakosti.

4.4 Svođenje nejednakosti na jednakosti – dodatne i dopunske varijable

- Svaka nejednakost kod koje je znak $\leq$ transformira se tako da se lijevoj strani doda *dopunska varijabla* i znak promijeni u $=$.
- Svaka nejednakost kod koje je znak $\geq$ transformira se tako da se lijevoj strani oduzme *dodatna varijabla* i znak promijeni u $=$.
- Jednakosti se ne diraju.

Primjer 4.8 Prevedite sljedeći linearni program u standardni oblik.

Maksimizirati $x_1 + x_2$

uz uvjete

$$\begin{aligned} x_1 + 2x_2 &\leq 3 \\ 4x_1 + 5x_2 &\geq 6 \\ 7x_1 + 8x_2 &= 15 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Rješenje:

Maksimizirati $x_1 + x_2$

uz uvjete

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 3 \\ 4x_1 + 5x_2 - x_4 &= 6 \\ 7x_1 + 8x_2 &= 15 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

4.5 Generiranje početnog mogućeg rješenja

Svakoj jednakosti koja ne sadrži dopunsku varijablu dodaj na lijevu stranu po jednu varijablu. Te dodane varijable nazivaju se *umjetne varijable*. Sve jednakosti će nakon toga sadržavati ili po jednu dopunsku varijablu ili po jednu umjetnu varijablu.

Ne-negativno početno rješenje dobiva se tako da se dopunske varijable postave na vrijednost desne strane odgovarajuće jednakosti, a sve ostale varijable na 0.

Primjer 4.9

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 3 \\ 4x_1 + 5x_2 - x_4 + x_5 &= 6 \\ 7x_1 + 8x_2 + x_6 &= 15 \end{aligned}$$

Ne-negativno rješenje ovog sustava je $x_3 = 3, x_5 = 6, x_6 = 15$, i $x_1, x_2, x_4 = 0$. Primjetimo da $x_1 = x_2 = 0$ rješenje nije moguće rješenje izvornog sustava.

4.6 Matrični oblik

Linearni program je u standardnom obliku ako su svi uvjeti prikazani u obliku jednadžbi i ako je poznato jedno moguće rješenje tog problema. U matričnom prikazu to izgleda ovako:

$$\begin{aligned} \text{optimizirati:} \quad & z = C^T X \\ \text{uz uvjet:} \quad & AX = B \\ \text{sa:} \quad & X \geq 0 \end{aligned}$$

gdje je X vektor-stupac nepoznanica koji uključuje sve dopunske, dodatne i umjetne varijable, C^T je vektor redak pripadajućih troškova, A je matrica koeficijenata u uvjetima, a B je vektor stupac desnih strana uvjeta.

4.7 Bazično rješenje

Ako promatramo sustav u matričnom obliku

$$\begin{aligned} \text{optimizirati:} \quad & z = C^T X \\ \text{uz uvjet:} \quad & AX = B \\ \text{sa:} \quad & X \geq 0 \end{aligned}$$

i ako označimo stupce matrice A dimenzije $m \times n$ s $A_1, A_2, \ldots, A_n$. Tada se matrica uvjeta $AX = B$ može napisati u vektorskom obliku kao

$$x_1 A_1 + x_2 A_2 + \cdots + x_n A_n = B \tag{4.2}$$

Vektori A i B su poznati m-dimenzionalni vektori. Potrebno je pronaći ne-negativna rješenja za $x_1, x_2, \ldots, x_n$. Pretpostavit ćemo da je $m \leq n$ i da je rang matrice A jednak m, što znači da postoji barem jedan m-torka A-vektora koji su linearno nezavisni.

Bazično moguće rješenje sustava 4.2 se dobiva postavljajući $n-m$ varijabli x na nulu, i pronalazeći ne-negativno rješenje za ostale x varijable, pod uvjetom da su m A-vektora koji odgovaraju varijablama koje nisu postavljene na nulu linearno nezavisni.

Varijable x koje nisu postavljene na nulu zovu se bazične varijable.

4.8 Troškovi umjetnih varijabli

Uvođenje dopunskih i dodatnih varijabli ne utiče na funkciju cilja te su stoga koeficijenti c_i uz svaku od tih varijabli jednaki 0. Međutim, umjetne varijable mijenjaju uvjete jer se dodaju samo na jednu stranu *jednakosti*, što znači da su jednakost bez umjetne varijable i jednost s dodanom uvjetnom varijablom iste samo ako je umjetna varijabla 0. Znači, umjetna varijabla mora u konačnom rješenju biti 0. Ona služi samo za lakše pronalaženje početnog rješenja.

Zbog toga se umjetne varijable u funkciju cilja uvrštaju s koeficijentom +M u problemu minimizacije te s koeficijentom -M u problemu maksimizacije. Ovdje +M predstavlja jako veliki pozitivni broj, a -M jako veliki negativni broj. Svako proizvođenje jedne jedinice umjetne varijable izrazito kvari funkciju cilja.

4.9 Razni primjeri

Primjer 4.10 Postavite MST u obliku linearnog programa.

Rješenje: Neka je zadana mreža predstavljena usmjerenim, težinskim grafom G=(V,E) gdje je V skup čvorova, a E skup lukova. Linearni program glasi:

$$\text{Minimizirati} \sum_{(i,j)\in E} c_{ij} x_{ij}$$

uz uvjete

$$\sum_{(i,j)\in E} x_{ij} = n - 1 \tag{4.3}$$

$$\sum_{(i,j)\in E(S)} x_{ij} \leq |S| - 1 \qquad \text{za svaki skup } S \subseteq V \text{ čvorova} \tag{4.4}$$

$$x_{ij} \geq 0 \text{ i cjelobrojni} \tag{4.5}$$

gdje je $E(S)$ skup grana koje imaju oba kraja u skupu čvorova S. Prvi uvjet izražava činjenicu da broj grana u konačnom rješenju mora biti $n-1$. Uvjet (4.4) zapravo predstavlja čitav niz uvjeta za svaki mogući $S \subseteq V$.

Ovi uvjeti sprečavaju postojanje ciklusa, tj. niti u jednom podgrafu induciranom s S ne smije postojati više od $|S| - 1$ grana. Ovih uvjeta ima eksponencijalno mnogo s obzirom na broj čvorova što bitno ograničava primjenljivost ovakve formulacije linearnog programa.

Primjer 4.11 U nekom gradu potrebno je bežičnim baznim stanicama osigurati pokrivanje šest naselja. Postavljanje baznih stanica moguće je na pet lokacija. Izgradnja bazne stanice na određenoj lokaciji omogućuje pristup nekima od gradova kao što je prikazano u tablici. Cijene izgradnje baznih stanica na lokacijama 1 do 5 su redom: 200, 100, 100, 300 i 500 tisuća eura. Na kojim lokacijama je potrebno izgraditi bazne stanice tako da bi se ostvario minimalni trošak izgradnje te da je svih šest naselja pokriveno? Postavite problem u obliku matematičkog programa.

	1	2	3	4	5
Naselje A	X				X
Naselje B		X	X		
Naselje C		X		X	X
Naselje D	X		X		
Naselje E				X	X
Naselje F		X		X	X

Rješenje: Definirajmo varijable odluke na sljedeći način:

$$x_i = \begin{cases} 1 & \text{ako je izgrađena stanica na lokaciji } i \\ 0 & \text{inače} \end{cases}$$

Cilj je

$$\text{minimizirati } 200x_1 + 100x_2 + 100x_3 + 300x_4 + 500x_5$$

uz uvjete

$$\begin{aligned} x_1 + x_5 &\geq 1 \\ x_2 + x_3 &\geq 1 \\ x_2 + x_4 + x_5 &\geq 1 \\ x_1 + x_3 &\geq 1 \\ x_4 + x_5 &\geq 1 \\ x_2 + x_4 + x_5 &\geq 1 \\ x_i \geq 0 &\text{ i cjelobrojno} \end{aligned}$$

Rješavanjem cjelobrojnog programa dobivamo dva optimalna rješenja. To su (1,0,1,1,0) i (1,1,0,1,0). Vrijednost funkcije cilja iznosi 600.

Primjer 4.12 Zadana je komunikacijska mreža i pripadajuća matrica ponuđenog prometa. (Matrica ponuđenog prometa za svaki par čvorova određuje promet koji se šalje od jednog do drugog čvora.) Zadani su kapaciteti i duljine grana. Odredite distribuciju prometa u mreži koja osigurava korištenje najkraćih putova, ali istovremeno minimizira najveću opterećenost grana. Problem postaviti kao linearni program.

Rješenje: Neka je mreža predstavljena usmjerenim, težinskim grafom $G = (V, E)$. Neka je K skup *uređenih* parova čvorova čiji ponuđeni promet je veći od 0 (tj. ti čvorovi imaju potrebu za slanjem podataka, s tim da se svaki smjer razmatra neovisno). Za svaki prometni zahtjev $k \in K$, definirajmo veličine d_k, s_k i t_k koje redom označavaju zahtijevanu brzinu prijenosa, početni čvor te odredišni čvor. Neka je α iskorištenje na grani koja ima najveće iskorištenje od svih grana u mreži. Neka su c_{ij} kapaciteti grana, a w_{ij} duljine grana. Linearni program glasi:

$$\text{minimizirati } \alpha + r \cdot \sum_{k \in K} \sum_{(i,j) \in E} w_{ij} x_{ij}^k$$

uz uvjete

$$\sum_{j:(i,j)\in E} x_{ij}^k - \sum_{j:(j,i)\in E} x_{ij}^k = 0, \qquad \text{za } k \in K, i \neq s_k, t_k \tag{4.6}$$

$$\sum_{j:(i,j)\in E} x_{ij}^k - \sum_{j:(j,i)\in E} x_{ij}^k = 1, \qquad \text{za } k \in K, i = s_k \tag{4.7}$$

$$\sum_{k\in K} d_k x_{ij}^k \leq c_{ij}\alpha, \qquad \text{za } (i,j) \in E \tag{4.8}$$

$$0 \leq x_{ij}^k \leq 1 \tag{4.9}$$

Konstanta r predstavlja vrlo malu vrijednost koja osigurava da minimizacija po α ima veći prioritet od minimizacije duljine putova.

Primjer 4.13 Problem sparivanja. Zadan je skup osoba od kojih treba složiti u parove za obavljanje nekog zadatka. Svaka osoba se izjasnila s kojima od drugih bi željela raditi u paru. Potrebno je složiti parove tako da se maksimalno poštuje želje osoba.

Primjer 4.14 Problem dodjeljivanja kad nema jednak broj poslova i ljudi.

4.10 Teorem dualnosti

Iz svakog linearnog programa da se konstruirati tzv. **dualni linearni program**. Dualni linearni program je s originalnim programom iz kojeg je nastao u uskoj vezi i iz ove povezanosti može se izvesti čitav niz interesantnih svojstava samog problema koji ova dva programa modeliraju. Konstrukciju dualnog programa obrazložit ćemo na primjeru, a kasnije ćemo ju formalizirati.

Neka je zadan sljedeći linearni program:

Maksimizirati $4x_1 + x_2 + 5x_3 + 3x_4$

uz uvjete

$$\begin{aligned} x_1 - x_2 - x_3 + 3x_4 &\leq 1 \\ 5x_1 + x_2 + 3x_3 + 8x_4 &\leq 55 \\ -x_1 + 2x_2 + 3x_3 - 5x_4 &\leq 3 \\ x_1, x_2, x_3, x_4 &\geq 0. \end{aligned}$$

Pokušajmo bez rješavanja programa pronaći jednu gornju granicu za z^*, tj. za optimalno rješenje programa. Ako pomnožimo drugi uvjet s $\frac{5}{3}$, dobivamo nejednakost

$$\frac{25}{3}x_1 + \frac{5}{3}x_2 + 5x_3 + \frac{40}{3}x_4 \leq \frac{275}{3}.$$

S obzirom na taj uvjet, svako optimalno rješenje će sigurno zadovoljavati sljedeće nejednakosti

$$4x_1 + x_2 + 5x_3 + 3x_4 \leq \frac{25}{3}x_1 + \frac{5}{3}x_2 + 5x_3 + \frac{40}{3}x_4 \leq \frac{275}{3}.$$

Odnosno, $z^* \leq \frac{275}{3}$. Bolju procjenu možemo dobiti ukoliko, na primjer, zbrojimo drugu i treću nejednakost. Dobivamo

$$4x_1 + 3x_2 + 6x_3 + 3x_4 \leq 58$$

što znači da je $z^* \leq 58$. Umjesto da nastavimo s pogađanjem gornje granice, možemo pristupiti na sljedeći način. Napravimo linearne kombinacije svih uvjeta, množeći i-ti uvjet nekim brojem y_i i zbrajajući sve uvjete. Dobivamo sljedeće:

$$(y_1 + 5y_2 - y_3)x_1 + (-y_1 + y_2 + 2y_3)x_2 + (-y_1 + 3y_2 + 3y_3)x_3 + (3y_1 + 8y_2 - 5y_3)x_4 \leq y_1 + 55y_2 + 3y_3. \tag{4.10}$$

Svaki y_i mora biti ne-negativan jer bi u suprotnom nejednakosti promijenile smjer. Nadalje, želimo da lijeva strana nejednakosti (4.10) bude gornja granica na $z = 4x_1 + x_2 + 5x_3 + 3x_4$. Ovo se postiže samo ako svaki koeficijent uz x_i iz (4.10) iznosi barem koliko i pripadajući koeficijent iz z. Dakle,

$$\begin{aligned} y_1 + 5y_2 - y_3 &\geq 4 \\ -y_1 + y_2 + 2y_3 &\geq 1 \\ -y_1 + 3y_2 + 3y_3 &\geq 5 \\ 3y_1 + 8y_2 - 5y_3 &\geq 3. \end{aligned}$$

Ako su koeficijenti y_i ne-negativni i ako zadovoljavaju prethodne četiri nejednakosti onda možemo zaključiti da svako moguće rješenje (x_1, x_2, x_3, x_4) zadovoljava nejednakost

$$4x_1 + x_2 + 5x_3 + 3x_4 \leq y_1 + 55y_2 + 3y_3.$$

U posebnom slučaju, to vrijedi i za optimalno rješenje,

$$z^* \leq y_1 + 55y_2 + 3y_3.$$

S obzirom da želimo pronaći minimalnu gornju granicu, postavljamo sljedeći linearni problem:

Minimizirati $y_1 + 55y_2 + 3y_3$

uz uvjete

$$\begin{aligned} y_1 + 5y_2 - y_3 &\geq 4 \\ -y_1 + y_2 + 2y_3 &\geq 1 \\ -y_1 + 3y_2 + 3y_3 &\geq 5 \\ 3y_1 + 8y_2 - 5y_3 &\geq 3 \\ y_1, y_2, y_3 &\geq 0. \end{aligned}$$

Ovaj program naziva se **dualni program** originalnog linearnog programa. S obzirom na ovaj dual, originalni program nazivamo **primarni** program.

Općenito, za primarni program

Maksimizirati

$$\sum_{j=1}^{n} c_j x_j$$

uz uvjete

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &\leq b_i \qquad (i = 1, 2, \ldots, m) \\ x_j &\geq 0 \qquad (j = 1, 2, \ldots, n) \end{aligned} \tag{4.11}$$

dualni program glasi:

Minimizirati

$$\sum_{i=1}^{m} b_i y_i$$

uz uvjete

$$\begin{aligned} \sum_{i=1}^{m} a_{ij} y_i &\geq c_j \qquad (j = 1, 2, \ldots, n) \\ y_i &\geq 0. \end{aligned} \tag{4.12}$$

Za svako moguće rješenje primarnog programa i svako moguće rješenje dualnog programa vrijedi

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i. \tag{4.13}$$

Kratko pisano, dokaz ovoga je sljedeći:

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{ij} y_i \Big) x_j = \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a_{ij} x_j \Big) y_i \leq \sum_{i=1}^{m} b_i y_i.$$

Izraz (4.13) ukazuje na sljedeće: ako nađemo moguće rješenje primarnog programa $(x_1^*, x_2^*, \ldots, x_n^*)$ i moguće rješenje duala $(y_1^*, y_2^*, \ldots, y_m^*)$ takvo da je

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$$

onda su oba ova rješenja optimalna. Iz (4.13) slijedi da svako moguće rješenje primarnog programa zadovoljava

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{i=1}^{m} b_i y_i^* = \sum_{j=1}^{n} c_j x_j^*$$

te da svako moguće rješenje duala zadovoljava

$$\sum_{i=1}^{m} b_i y_i \geq \sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

Veza između primarnog i dualnog problema izrekli su Gale, Kuhn i Tucker 1951. godine kroz sljedeći teorem.

Teorem 4.1 (Teorem dualnosti) Ako primarni program (4.14) ima optimalno rješenje $(x_1^*, x_2^*, \ldots, x_n^*)$, onda dualni program (4.12) ima optimalno rješenje $(y_1^*, y_2^*, \ldots, y_m^*)$ takvo da vrijedi

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

4.10.1 Smisao dualnih varijabli

Za mnoge LP programe iz prakse, varijable y_i dualnog problema možemo smisleno interpretirati. Promotrimo sljedeći program.

Maksimizirati

$$\sum_{j=1}^{n} c_j x_j$$

uz uvjete

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &\leq b_i \qquad (i = 1, 2, \ldots, m) \\ x_j &\geq 0 \qquad (j = 1, 2, \ldots, n). \end{aligned} \tag{4.14}$$

Prisjetimo se primjera ?? s proizvodnjom namještaja. Svaki x_j izražava količinu proizvedenih proizvoda j, dok b_i izražava raspoloživ resurs i. Svaki a_{ij} je izražen u jedinicama resursa i po jedinici proizvoda j, tj. a_{ij} je količina resursa i potrebna za proizvodnju jedinice proizvoda j. Vrijednosti c_j označavaju neto profit po jediničnoj količini proizvoda j. Pisano u obliku mjernih jedinica, primarni program glasi:

Maksimizirati

$$\sum_{j=1}^{n} (\text{profit po jedinici proizvoda } j)(\text{količina proizvoda } j) = \text{ukupni profit}$$

uz uvjete

$$\begin{aligned}\sum_{j=1}^{n}\binom{\text{količina resursa } i}{\text{po jedinici proiz. } j}(\text{količina proiz. } j) &\leq (\text{količina resursa } i) \qquad i=1,2,\ldots,n\\ \text{količina proizvoda } j &\geq 0 \qquad j=1,2,\ldots,n.\end{aligned}$$

Dualni program glasi:

Minimizirati

$$\sum_{i=1}^{m}\binom{\text{ukupna količina}}{\text{resursa } i}\binom{\text{jedinična vrijednost}}{\text{resursa } i} = \text{ukupna vrijednost svih resursa}$$

uz uvjete

$$\begin{aligned}\sum_{i=1}^{n}\binom{\text{količina resursa } i}{\text{po jedinici proiz. } j}\binom{\text{jedinična vrijednost}}{\text{resursa } i} &\geq \binom{\text{profit po jednici}}{\text{proizvoda } j} \qquad j=1,2,\ldots,n\\ \text{jedinična vrijednost resursa } i &\geq 0 \qquad i=1,2,\ldots,m\end{aligned}$$

Sljedeći teorem pojašnjava smisao dualnih varijabli.

Teorem 4.2 Ako primarni program (4.14) ima barem jedno nedegenerirano bazično optimalno rješenje onda postoji pozitivan ϵ takav da vrijedi sljedeće: Ako je $|t_i| \leq \epsilon$ za svaki $i = 1, 2, \ldots, m$ tada problem

Maksimizirati

$$\sum_{j=1}^{n} c_j x_j$$

uz uvjete

$$\begin{aligned}\sum_{j=1}^{n} a_{ij}x_j &\leq b_i + t_i \qquad i=1,2,\ldots,m\\ x_j &\geq 0 \qquad j=1,2,\ldots,n\end{aligned}$$

ima optimalno rješenje i vrijednost funkcija cilja tog optimalnog rješenja iznosi

$$z^* + \sum_{i=1}^{m} y_i^* t_i$$

gdje je z^* optimalno rješenje primarnog programa (4.14), a $y_1^*, y_2^*, \ldots, y_m^*$ čine optimalno rješenje dualnog programa od (4.14).

Dakle, sa svakim jediničnim povećanjem raspoložive količine resursa i, profit se može povećati za y_i^*. U primjeru s proizvodnjom namještaja, neki od resursa su do kraja iskorišteni, dok su drugi resursi neiskorišteni. Kad bi proizvođač bio u mogućnosti kupiti još resursa, logično je da ne bi imalo smisla kupovati resurse koji nisu u potpunosti iskorišteni. Povećanje količine takvih resursa ne može povećati profit. Međutim, kad bi proizvođač mogao nabaviti dodatne količine resursa koji su u potpunosti iskorišteni, to bi mu povećalo profit i to točno za odgovarajući y_i^* iznos. Vrijednost y_i^* naziva se **marginalna vrijednost** resursa i jer označava koliko proizvođaču vrijedi dodatna jedinična količina resursa i.

Primjer 4.15 Dvije vrste materijala su na raspolaganju za proizvodnju stolova i klupa, hrastovina i borovina. Na raspolaganju je 150m^2 hrastovine, 100m^2 borovine i 80 radnih sati. Stol zahtijeva 5m^2 hrastovine, 2m^2 borovine i 4 sata rada. Klupa zahtijeva 2m^2 hrastovine, 3m^2 borovine i 2 sata rada. Profit po stolu je 12 eura, a po stolcu 8 eura. Maksimizirati profit.

Linearni program glasi:

Maksimizirati

$$f = 12x_1 + 8x_2$$

uz uvjete

$$\begin{aligned} 5x_1 + 2x_2 &\leq 150 \\ 2x_2 + 3x_2 &\leq 100 \\ 4x_1 + 2x_2 &\leq 80 \\ x_1, x_2 &\geq 0. \end{aligned}$$

Nakon svođenja uvjeta na jednakosti, uvedene su dopunske varijable x_3, x_4 i x_5. Optimalno rješenje proširenog programa je $(5, 30, 65, 0, 0)$ te je $f^* = 300$.

Literatura

[1] Kalpić, Mornar, Fertalj. Predavanja iz kolegija Algoritmi i strukture podataka.

[2] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[3] R.K.Ahuja, T.L.Magnanti, and J.B.Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, 1993.

[4] T.H.Cormen, C.E.Leiserson, R.L.Rivest. *Introduction to Algorithms.* Mc-Graw Hill, 1999.

[5] V.K.Balakrishnan. *Graph Theory.* Schaum's Outlines. Mc-Graw Hill, 1997.