Grafovi - problem uparivanja

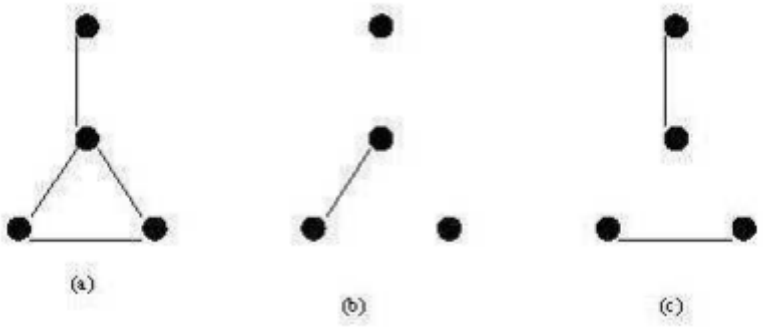
Za zadati neusmereni graf G=(V,E) uparivanje je skup disjunktnih grana (grana bez zajedničkih čvorova).
Savršeno uparivanje je uparivanje u kome su ovi čvorovi upareni (pripadaju nekoj od grana iz uparivanja).
Maksimum (optimalno) uparivanje je uparivanje sa max brojem grana (tj. |M| >= |M'| za bilo koje uparivanje M' u G).
Maksimalno uparivanje je uparivanje koje se ne može proširiti dodavanjem nove grane (tj. M nije podskup od M' za bilo koje uparivanje M' u G).

Teorema 1: Ako M je maksimum uparivanje M => M je maksimalno uparivanje
Dokaz: Pretpostavimo suprotno tj. da M nije maksimalno
Onda postoji uparivanje M' u G tako da M podskup od M'

- ⇨ |M| < |M'|
- ⇨ M nije maksimum uparivanje
- ⇨ kontradikcija

Obrnuto tvrđenje ne važi. Na primer, u grafu sa slike a), maksimalno uparivanje nije maksimum uparivanje.

a) polazni graf
b) maksimalno uparivanje
c) maksimum uparivanje



Maksimalno uparivanje M za dati graf G se može dobiti sledećim gramzivim algoritmom:

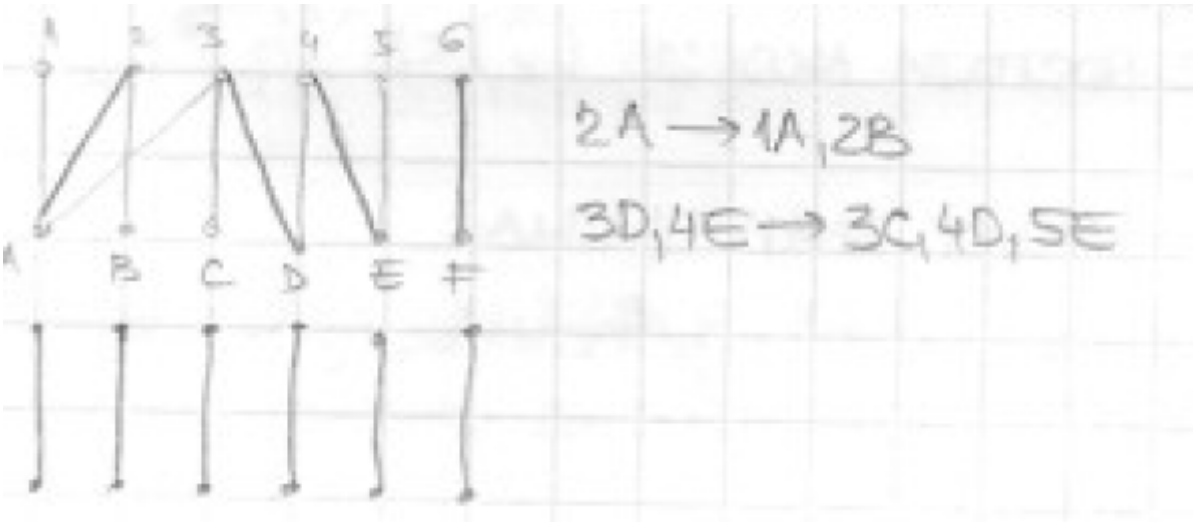
Maximal Matching(G, V, E)
1. $M = \phi$
2.While(no more edges can be added)
    2.1 Select an edge,e,which does not have any vertex in common with edges in M
    2.2 $M = M \cup e$
3. return$M$

Za neusmereni graf kod koga je |V| = 2n i stepen svakog čvora bar *n* postoji savršeno uparivanje.

Bipartitivni graf je graf čiji se čvorovi mogu podeliti na 2 disjunktna podskupa tako da u grafu postoje samo grane između čvorova iz različitih podskupova.

Neka je G=(U, V, E) bipartitivni graf u kome su U i V disjunktni skupovi čvorova, E je skup grana koje povezuju neke čvorove iz U sa nekim čvorovima iz V. Naći uparivanje sa max brojem grana u bipartitnom grafu G.

Formiramo parove u skladu sa nekom strategijom i dobijamo maksimalno uparivanje. Možemo li ga nekako popraviti?



Alternirajući put P za dato uparivanje M je put od neuparenog čvora v∈ V do neuparenog čvora u ∈ U, pri čemu su grane naizmenično u E\M, odnosno M, tj. prva grana ne pripada M, druga pripada,...
Alternirajući putevi su ti koji omogućavaju povećanje uparivanja.
Važi: uparivanje je optimalno akko nema alternirajućih puteva.

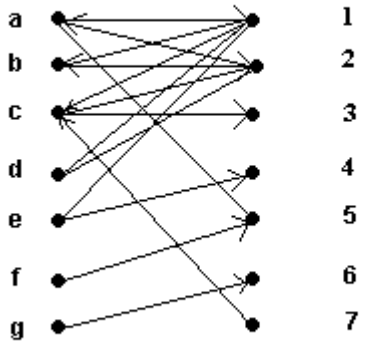
Kako pronaći alternirajući put?
Neusmereni graf transformišemo u usmereni graf G' tako što usmeravamo grane iz M od U do V, a grane iz E\M od V do U. Alternirajući put tada odgovara usmerenom putu od neuparenog čvora u V do neuparenog čvora u U, a on se može naći pomoću DFSa.
Kako je složenost pretrage O(|V|+|E|), onda je složenost algoritma O(|V| (|V|+|E|)). Poboljšani algoritam je $O((\sqrt{|V|}(|V|+|E|)))$

1. Nalaženje optimalnog uparivanja u grafu sa 2n čvorova ako svi čvorovi imaju stepen bar n
Neka je G = (V, E) neusmereni graf kod koga je |V| = 2n i stepen svakog čvora je bar n. Koristićemo indukciju po veličini uparivanja m. Bazni slučaj m = 1 rešava se formiranjem uparivanja veličine jedan od proizvoljne grane grafa. Pokazaćemo da se proizvoljno uparivanje koje nije savršeno može proširiti ili dodavanjem jedne grane ili zamenom jedne grane dvema novim granama. U oba slučaja povećava se veličina uparivanja za jedan.
Posmatrajmo uparivanje M u grafu G sa m grana, pri čemu je m < n. Najpre proveravamo sve grane van upraviranja M da bismo proverili da li se neka od njih može dodati u M. Ako nađemo takvu granu, problem je rešen - nađeno je veće uparivanje. U protivnom je M maksimalno uparivanje. Ako M nije savršeno uparivanje, postoje bar dva neuparena čvora v1 i v2. Iz ta dva čvora po pretpostavci izlaze najmanje 2n grana. Sve te grane vode ka uparenim čvorovima (u protivnom bi se u uparivanje M mogla dodati grana, a to bi bilo superotno pretpostavci da je M maksimalno). Pošto u M ima manje od n grana, a iz v1 i v2 izlazi najmanje 2n grana, u uparivanju M postoji grana (u1, u2) koja je susedna sa bar tri grane iz v1 i v2. Pretpostavimo bez smanjenja opštosti da su to grane (u1, v1), (u1, v2) i (u2, v1). Dodavanjem tih grana u M a brisanjem grane (u1, u2) se dobija veće uparivanje.
Opisani algoritam je još jedan primer pohelpnog algoritma

2. Nalaženje optimalnog uparivanja u bipartitnom grafu
Neka je G=(U, V, E) bipartitivni graf u kome su U i V disjunktni skupovi čvorova a E skup grana koje povezuju neke čvorove iz U sa nekim čvorovima iz V.
Pretpostavimo da polazimo do nekog uparivanja koje može biti maksimalno ali ne i optimalno. Cilj nam je da povećamo broj uparenih čvorova. Alternirajući put R za dato uparivanje M je put od neuparenog čvora v iz V do neuparenog čvora u iz U, pri čemu su grane puta R naizmenično u E\M, odnosno M. Drugim rečima, prva grana puta R (v, w) ne pripada M, a druga grana (w, h) pripada M i tako dalje do poslednje grane (z, u) puta P koja ne pripada M. Broj grana na putu R mora biti neparan, jer put polazi iz V a završava se u U.
Teorema: Uparivanje je optimalno akko nema alternirajućih puteva.
Teorema o alternirajućim putevima direktno sugeriše algoritam. Započinjemo sa pohlepnim algoritmom, dodajući grane u uparivanje sve dokle je to moguće. Onda prelazimo na traženje alternirajućih puteva i povećavamo uparivanje sve do trenutka kada više ne bude alternirajućih puteva. Dobijeno uparivanje je tada optimalno.
Kako naći alternirajuće puteve?
Transformišemo neusmereni graf G=(U, V, E) u usmereni graf G' usmeravajući grane iz M od U ka V. Alternirajući

put u G tada odgovara usmerenom putu od neuparenog čvora u V do neuparenog čvora u U. Takav usmereni put se može pronaći bilo kojim postupkom obilaska grafa, npr. DFS. Složenost obilaska je O(|V|(|V| + |E|)).
Poboljšanje: Moguće je traženje više alternirajućih puteva jednom pretragom. Potrebno je osigurati da ovi putevi ne menjaju jedan drugog. Jedan način da se obezbedi nezavisnost takvih alternirajućih puteva je uvođenje ograničenja da skupovi čvorova na pojedinačnim putevima budu disjunktni. Poboljšani algoritam za nalaženje alternirajućih puteva bi bio: primenjujemo BFS na G' od skupa neuparenih čvorova u V, nivo po nivo, do nivoa u kome su pronađeni neupareni čvorovi iz U; zatim iz grafa indukovanog pretragom u širinu vadimo maksimalni skup disjunktnih puteva u G', kojima odgovaraju alternirajući putevi u G; to se izvodi pronalaženjem prvog puta, uklanjanjem njegovih čvorova, pronalaženjem novog puta ... biramo maksimalni skup, da bismo posle pretrage dobili što veće uparivanje; svaki novi disjunktni put povećava uparivanje za jednu granu; na kraju povećavamo uparivanje korišćenjem pronađenog skupa disjunktnih puteva; proces se nastavlja sve dokle je moguće pronaći alternirajuće puteve tj. dokle je neupareni čvor iz V dostižan iz nekog neuparenog čvora u U.
Složenost:Ukupna složenost poboljšanog algoritma je O($\sqrt{V}$ (|V|+|E|) .

3. Dat je bipartitni graf G = (V,U,E) sa skupovima čvorova V = {a, b, c, d, e, f, g} i U = {1, 2, 3, 4, 5, 6, 7} i skupom grana E = {(a, 1), (a, 2), (a, 5), (b, 1), (b, 2), (c, 1), (c, 2), (c, 3), (c, 7), (d, 1), (d, 2), (e,1), (e, 4), (f, 5), (g, 6)}. Naći optimalno uparivanje u ovom grafu, počevši od uparivanja:
a) M_0 = {(a, 1), (b, 2), (c, 3), (e, 4), (f, 5), (g, 6)}.
b) M_0 = {(a, 5), (b, 2), (d, 1)}.



Rešenje:
a) Na slici je prikazano da zbog pronalaska alternirajuceg puta u G, usmeravamo grane G tako da one koje su u uparivanju M_0 budu udesno, a one koje nisu orjentisane su ulevo. Neupareni su čvorovi 7, d i pokušavamo da tražimo put od neuparenog čvora 7 do neuparenog *d*.
Međutim on ne postoji – iz 7 idemo u *c*, dalje moramo do 3 zbog uparivanja M_0 , a iz 3 nema dalje
=> M_0 je optimalno uparivanje.

Može se do uparivanja M doći i ako se kao polazno uparivanje uspostavi uparivanje N={(a,5), (b,2), (c,3), (e,4), (g,6)}, jer bi se uočio alternirajući put f, 5, a.

4. Dat je bipartitivni graf *G* čiji su čvorovi *a1, a2, a3, a4* odnosno *b1, b2, b3,b4* i čiji skup grana je {(a1, b2), (a1,b3), (a2,b1), (a2,b3), (a2,b4), (a3,b1), (a3,b3), (a3,b4), (a4,b4)}. Naći optimalno uparivanje u ovom grafu, počevši od uparivanja M={(a1,b3), (a2,b4), (a3,b1)}. Da li je to uparivanje potpuno?

Rešenje:
1. Počnimo sa neuparenim čvorom a4.
2. Uočimo granu (a4,b4)∈ E\M
3. Nastavimo gradnju alternirajućeg puta do čvora a2, jer (a2,b4) ∈ M
4. Uočimo granu (a2,b1)∈ E\M
5. Nastavimo gradnju alternirajućeg puta do čvora a3, jer (a3,b1) ∈ M
6. Uočimo granu (a3,b3)∈ E\M
7. Nastavimo gradnju alternirajućeg puta do čvora a1, jer (a1,b3) ∈ M
8. Uočimo granu (a1,b2)∈ E\M

Pronašli smo alternirajući put od neuparenog čvora a4 do neuparenog čvora b2.
Zato sad pravimo izmene tj. pravimo novo uparivanje M1={(a1,b2), (a2,b1), (a3,b3), (a4,b4)}.
Imamo uparene sve čvorove. Broj grana u sparivanju M1 je jednak broju čvorova 4 u oba disjunktna skupa, te sada imamo i maksimalno uparivanje i u suštini i potpuno uparivanje.

5. Petoro osoba (Aleksa, Branko, Vlada, Goran, Dragan) traže posao. Na tržištu su u ponudi poslovi A, B, C, D, E. Aleksa je kvalifikovan za poslove C, E. Branko je kvalifikovan za poslove A, D. Vlada je kvalifikovan za poslove C, E. Goran je kvalifikovan za posao D. Dragan je kvalifikovan za poslove B, E.
a) Pronađite uparivanje koje će većini ljudi obezbediti posao za koji su kvalifikovani.
b) Da li će svi dobiti posao?
c) Može li se pronaći alternirajući put za nađeno uparivanje pod a) ?

Rešenje:
Postoji savršeno uparivanje (Aleksa, C), (Branko, A), (Vlada, E), (Goran, D), (Dragan,B).

6. Pronaći uparivanje sa maksimalnim brojem grana za bipartitivni graf G=(V,E) gde
V={{a, b, c, d}, {1, 2, 3, 4}}, E={ (a,1), (a, 2), (b, 1), (b, 4), (c, 1), (c, 2), (c, 3), (d, 1), (d, 3), (d, 4)}. Da li je dobijeno uparivanje savršeno?

Rešenje:
Postoji i optimalno i savršeno uparivanje: d->4<-b->1<-a->2<-c->3, tj. M={(a,2), (b,1), (c,3), (d,4)}
POSTUPAK DOBIJANJA DATOG UPARIVANJA: Pretpostaviti da skup M je prazan. Zato
1. Upariti **a->1**
2. kako nijedna grana nije uparena sa **b**, upariti b->1, vratite se na **a** i uparite sa **2**, tj. **b->1<-a->2**
3. kako nema uparivanja za **c**, upariti **c** sa **2**, vratiti se na **a**, upariti **a** sa **1**, vratiti se na **b**, upariti **b** sa **4**, tj. **c->2<-a->1<-b->4**
4. kako ne postoji uparivanje za **d**, preslikati **d** u **4** i vratite se prethodnom putanjom, zatim preslikati **c** u **3**, **b** u **1**, **a** u **2**, što daje uparivanje **d->4<-b->1<-a->2<-c->3**
Ovo uparivanje je savršeno.

7. Šest prijatelja idu na maskenbal. Svaki od njih želi da izabere različit kostim. Emiliji se dopada kostim klovna, Boži se dopada kostim duha i Frankeštajna. Kosti se dopada kostim klovna i kauboja. Danilu se dopadaju kostimi duha, Drakule i Frankeštajna. Jovanu se dopadaju kostimi klovna, kauboja i Šerlok Holmsa. Filipu se dopadaju kostimi Drakule i klovna.
Da li je moguće da svaka osoba dobije kostim koji joj se dopada?

9. Zadat je težinski bipartitni graf G = (V,E) sa *n* čvorova i *m* grana. **Kritična težina uparivanja M** (u grafu G) je težina najteže grane u uparivanju M. Konstruisati algoritam složenosti $O(\sqrt{n}(m+n)\log m)$ za nalaženje **optimalnog uparivanja sa minimalnom kritičnom težinom.** (uporediti sa zadatkom o praking mestima)

Rešenje:
Koristićemo kombinaciju binarne pretrage i algoritma za nalaženje optimalnog uparivanja.
Rešavamo najpre malo drugačiji problem: pitamo se da li za dato *x* postoji optimalno uparivanje takvo da su težine svih njegovih grana manje ili jednake od *x*
Ovaj problem možemo rešiti tako što iz grafa uklonimo sve grane čije su težine veće od *x* i zatim proveravamo da li optimalno uparivanje u novodobijenom smanjnom grafu ima jednak broj grana kao optimalno uparivanje u polaznom grafu.
Složenost ove provere jednaka je složenosti algoritma za nalaženje optimalnog uparivanja u grafu koje je $O(\sqrt{n}(m+n))$.
U grafu G ima *m* grana, te ima najviše *m* različitih težina. Binarnom pretragom tražimo najmanje *x* tako da je *x* težina neke grane i da postoji optimalno uparivanje u kome sve grane imaju težinu manju ili jednaku *x.* Zato je ukupna složenost $O(\sqrt{n}(m+n)\log m)$.

9. Na parkingu postoji mnogo automobila i parking mesta, a svi automobili žele da dospeju do parking mesta. Zbog saobraćajnih propisa, automobil može da se vozi samo paralelno sa granicama parkinga i samo brzinom od jednog kvadrata po jedinici vremena. Obično svi automobili voze na najbliže mesto za parkiranje, ali to može da se ispostavi loše za neke automobile. Razmotrite, na primer, sledeće park automobila
.C.....P.X...
XX.......X..P
XX.....C.....
('C' je oznaka za automobil, 'P' za parking mesto, 'X' za zid, '.' za prazno mesto)

Objašnjenje: Ukoliko automobil pozicioniran krajnje dole vozi *do svog najbližeg parking mesta*, onda gornji levi automobil mora voziti skroz desno, uzimajući 13 jedinica vremena. Ako, međutim, automobil pozicioniran krajnje dole vozi *do parking mesto desno*, biće potrebno 6 jedinica vremena za oba automobila da pronađu mesto za parkiranje.
Vaš program mora da vrati minimalnu količinu vremena koje je potrebna da svaki automobil dobije mesto za parkiranje (pod pretpostavkom da se automobili ponašaju na društven odgovoran način opisan u gornjem objašnjenju). Svi automobili startuju sa praznog mesta. Automobili su dovoljno mali i bilo koji broj automobila može da vozi na istom kvadratu parkinga istovremeno. Oni mogu da voze preko praznih mesta i parking mesta, ali ne kroz zidove. Svaki automobil mora da završi na zasebnom parking mestu.
Ako je nemoguće da se svaki automobil doveza na parking mesto, vratite -1.

Ograničenja
- broj elemenata parkinga [1..50]
- svi elementi parkinga su iste duzine
- svaki element parkinga je duzine [1..50]
- svaki karakter na mapi parkinga je 'C', 'P', 'X' ili '.'.
- broj automobila je ne veci od 100, kao i broj parking mesta.

Test primeri
1)
{"C.....P",
 "C.....P",
 "C.....P"}
Rezultat: 6
Svaki automobil vozi bas do parking mesta koje mu je naspramno
2)
{"C.X.....",
 "..X..X..",
 "..X..X..",
 ".....X.P"}
Rezultat: 16
Potrebna je slalom voznja i 16 jedinica vremena za ovaj automobil.
3)
{"XXXXXXXXXXX",
 "X......XPPX",
 "XC...P.XPPX",
 "X......X..X",
 "X....C....X",
 "XXXXXXXXXXX"}
Rezultat: 5
Inace, potrebno je 11 umesto 5 jedinica vremena ako automobil krajnje dole vozi do svog najblizeg parking mesta.
4)
{".C.",
 "...",
 "C.C",
 "X.X",
 "PPP"}
Rezultat: 4

Dok voze, automobili mogu boraviti na istom praznom mestu ili parking mestu, ali moraju da se na kraju parkiraju na razlicita parking mesta.

5)
{"CCCCC",
 ".....",
 "PXPXP"}
Rezultat: -1
Nema dovoljno parking mesta za sve automobile.

6)
{"..X..",
 "C.X.P",
 "..X.."}
Rezultat: -1
Automobil ne moze da dodje do parking mesta.

Resenje:
U ovom problemu moramo UPARITI svaki automobil sa parking mestom.
Dodatni uslov je minimizovati potrebno vreme da svaki auto nadje svoje parking mesto.
Moracemo da konstruisemo bipartitni graf: skup cvorova A cine automobili, skup cvorova B cine parking mesta.
Svaka grana koja povezuje cvorove iz skupa A, B ima cenu (ali ne i kapacitet!) potrebnog vremena parkiranja. Ako parking mesto nije doseznoe, mozemo ili grani dodeliti beskonacnu vrednost ili je ukloniti.

Ove cene cemo utvrditi pomocu BFS pretrage.

Pretpostavimo da ocekivani rezultat je manji ili jednak od konstante D. Onda, postoji uparivanje u kom svaka grana koja spaja automobil i parking mesto ima cenu manju ili jednaku od D.
Otuda, uklanjanje grana čije cene su veće od D neće imati uticaj na rešenje problema.

Dakle, to svojstvo nam sugerise binarnu pretragu za vrednost D, uklanjajući sve grane sa cenom većom od D, i potom sprovođenje algoritma za traženje maximum bipartitno-uparivanje. Ako postoji uparivanje u kom se svaki auto moze dovesti do parking mesta, onda možemo smanjiti D. U suprotnom, moramo povećati D.

Međutim postoji i brže i elegantnije rešenje koristeći pretragu prvog prioriteta. Umesto da se vrednost D fiksira i čuva, možemo da odaberemo strategiju: povećaj D kad god se utvrdi da mu je vrednost premala.
Počećemo sa D = 0.
Potom ćemo imati iteraciju kroz svaki automobil i pokušati da nađemo povećavajući put u kom nijedna grana nema cenu veću od D. Ako takvog puta nema, povećavamo D dok ne nađemo opisani put.
Jasno je da ćemo D zapravo uvećati za najmanju moguću vrednost. Da bismo to postigli, mi ćemo tražiti povećavajući put najmanje cene. Cena puta se nalazi kao maximum cena grana na tom putu. Ovakav povećavajući put ćemo naći koristeći pretragu prvog prioriteta koja je slična algoritmu pretraga prvog prioriteta povećavajučeg puta koji je predstavljen u udžbeniku.

Sledi C++ kod rešenja:

```
#define DIM 105
 struct node {
  int where, cost, from;
  node(int _where, int _cost, int _from): where(_where),
cost(_cost), from(_from) {};
};

bool operator < (node a, node b) {
  return a.cost > b.cost;
}
```

```
int minTime(vector  park)
{
  // kreiramo matricu cena cost[i][j] = cena voznje aitomobila i do parking mesta j, putem BFS pretrage
  // cvorovi 0, 1, ..., N - 1 su oznake automobila
// cvorovi, N + 1, ..., N + M - 1 su oznake parking mesta
// N + M je super-ponor grafa
  int D = 0, sink = N + M;
  int car_match[DIM], park_match[DIM];
  memset(car_match, -1, sizeof(car_match));
  memset(park_match, -1, sizeof(park_match));

  for (int source = 0; source < N; ++ source) {
    bool visited[2*DIM];
    memset(visited, false, sizeof(visited));
    int from[2*DIM];
    memset(from, -1, sizeof(from));
    priority_queue pq;
    pq.push(node(source, 0, -1));
    while (!pq.empty()) {
      int cst = pq.top().cost, where = pq.top().where,
_from = pq.top().from;
      pq.pop();
      if (visited[where]) continue;
      visited[where] = true;
      from[where] = _from;
      // ako where oznacava automobil tj where je 0..N-1, pokusaj svih M parking mesta
      if (where < N) {
        for (int i = 0; i < M; ++ i) {
          // ako grana ne postoji ili taj auto je vec uparen sa tekucim parking mestom
          if (cost[where][i] == infinity || car_match[where] == i) continue;
          int ncst = max(cst, cost[where][i]);
          // i-to parking mesto je N + i
          pq.push(node(N + i, ncst, where));
        }
      }
      else {
        // ako to parking mesto nije upareno, nasli smo povecavauci put minimalne cene
        if (park_match[where - N] == -1) {
          from[sink] = where;
          // ako je potrebno da se poveca D, povecajmo ga
          D = max(D, cst);
          break;
        }
        // u suprotnom, pratimo granu povratka
        int next = park_match[where - N];
        int ncst = max(cst, cost[next][where]);
        pq.push(node(next, ncst, where));
      }
    }

    int where = from[sink];
    //ako nije pronadjen povecavajuci put, nema resenja
    if (where == -1)
      return -1;
    // pratimo povecavajuci put
    while (from[where] > -1) {
      int prev = from[where];
```

```
    // ako where je parking mesto,  grana (prev, where) je forward grana i uparivanje mora da se azurira
    if (where >= N) {
      car_match[prev] = where;
      park_match[where - N] = prev;
    }
    where = prev;
   }
  }

  return D;
}
```

10. Vi ste student prodekan fakulteta M. Uslovi diplomiranja na M su donekle komplikovani. Svaki uslov se odnosi na činjenicu da student mora da uzme neki broj predmeta iz nekog skupa, a svi uslovi moraju biti ispunjeni za studenta ako želi da diplomira. Svaka predmet je predstavljen jednim karakterom. Na primer, jedan zahtev može da bude "Uzmite bilo koja 2 predmeta od B, C, D, ili F." Ono što komplikuje stvari je činjenica da nijedan predmet ne može da se iskoristi da bi se zadovoljilo više od jednog uslova. I tako da studenti dolaze do Vas svo vreme ošamućeni i zbunjeni, jer oni ne znaju koliko su blizu zadovoljavanja uslova s kojim mogu diplomirati!
Svaki predmet je predstavljena kao poseban znak čija je ASCII vrednost je između 32 i 126, uključujući 32 i 126, ali nije numerički karakter ('0 '- '9'). Kao ulazne parametre dobijate String prIzbor, koji predstavlja izabrane predmete jednog studenta. Takođe ćete dobiti na ulazu String [] uslovi, koji predstavlja spisak uslova neophodnih za sticanje diplome. Svaki član spiska će početi pozitivnim celim brojem, a zatim slede oznake predmeta. Na primer, uslov " Uzmite bilo koja 2 predmeta od B, C, D, ili F " će biti predstavljen u spisku kao "2BCDF".

Konstruisati metod koji treba da vrati string (u ASCII poretku) koji predstavlja predmete koje student treba da uzme u cilju da diplomira. Predmeti se ne mogu uzeti više od jednog puta. Ako postoji više od jednog skupa predmeta koji će omogućiti studentu da diplomira, vratite najmanji skup. Ako postoji više najmanjih skupova, vratite leksikografski najmanje rešenja. Konačno, ako ne postoji skup koji će omogućiti da student diplomira, vratite "0".
Ograničenja
- prIzbor će biti dužine [0..50] karaktera
- nece biti duplih predmeta u prIzbor
- uslovi sadrze [1.. 50] elemenata
- svaki clan spiska uslova sadrzi pozitivan ceo broj bez vodećih nula između [1..100] nakon kog slede oznake predmeta
- svaki član spiska uslova će biti dužine [1..50] karaktera

Test primeri
1)
"A"
{"2ABC","2CDE"}
Rezultat: "BCD"
Student mora uzeti 2 predmeta iz {A,B,C}, i 2 predmeta iz {C,D,E}. On je vec uzeo predmet A.

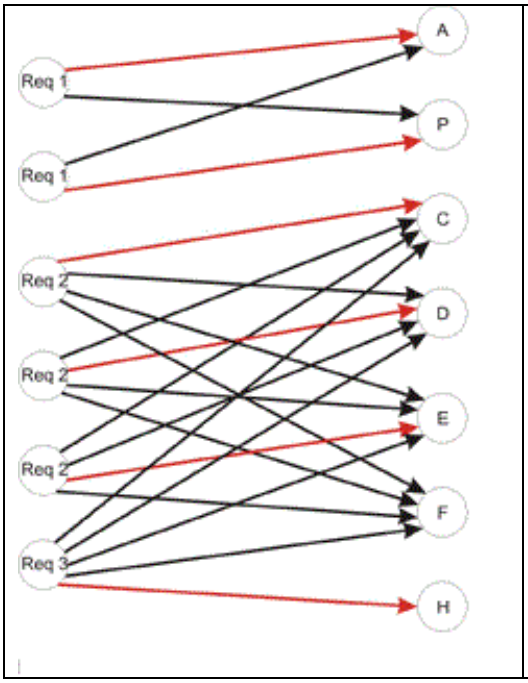
2)
"+/NAMT"
{"3NAMT","2+/","1M"}
Rezultat: ""
Student je vec ispunio sve uslove – cestitajte mu

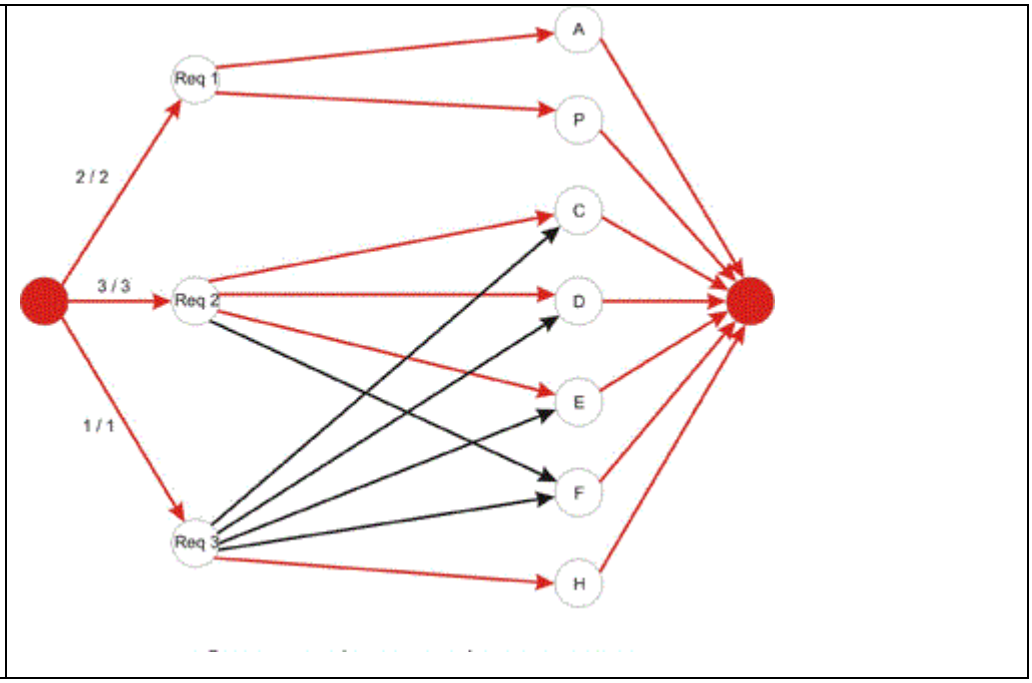
3)
 "A"
{"100%*Klju"}
Rezultat: "0"
Ne mozete odabrati 100 predmeta od ponudjenih 6.

4)
""

{"5ABCDE","1BCDE,"}
Rezultat: ",ABCDE"

5)
"CDH"
{"2AP", "3CDEF", "1CDEFH"}
Rezultat: "AEP"

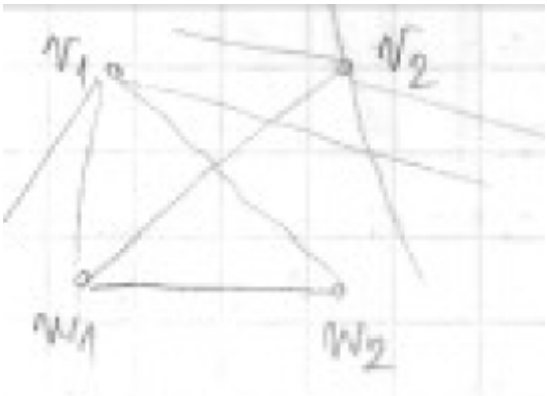




11. Konstruisati algoritam linearne složenosti za određivanje optimalnog uparivanja u stablu.
Rešenje:
Uzmemo proizvoljan list *v* i uparujemo ga sa svojim ocem *w*, uklanjamo oba čvora iz stabla (zajedno sa ostalim sinovima koji ostaju neupareni). Po induktivnoj hipotezi umemo da rešimo preostali problem.

Pokažimo da grana (v,w) zaista pripada nekom optimalnom uparivanju. Ako optimalno uparivanje ne sadrži granu (v,w), onda čvor v nije uparen (jer je povezan jedino sa w).
Ako je grana (u,w) u optimalnom uparivanju, možemo je zameniti granom (v,w) i opet dobijamo optimalno uparivanje koje sadrži granu (v,w).

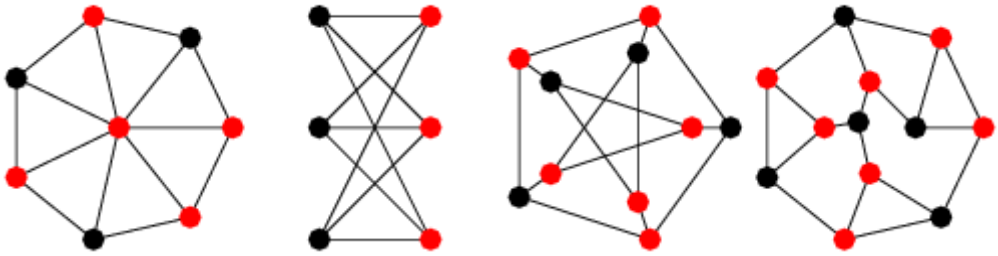
12. Konstruisati algoritam složenosti $O(|V|^2)$ za nalaženje savršenog uparivanja u vrlo gustom grafu, tj. grafu sa 2n čvorova takvom da za svaka dva njegova nesusedna čvora *u* i *v* važi: $d(u) + d(v) \geq 2n$.
Rešenje:
Najpre se formira maksimalno uparivanje i lista neuparenih čvorova. Ako uparivanje nije savršeno, uzimamo dva neuparena čvora v1, v2. S obzirom da ova grana nije dodata uparivanju, oni nisu susedni, tj, važi $d(v1) + d(v2) \geq 2n$. i tražimo granu uparivanja (w1,w2) tako da od v1, v2 ka w1, w2 vode bar tri grane (ovo je složenosti O(|V|)) i menjajući granu (w1, w2) sa dve nove od v1, v2 ka w1, w2.



Pretpostavimo suprotno da je svaka od grana uparivanja susedna sa max 2 grane iz v1, v2 – tada pošto uparivanje nije maksimalno – u njemu postoji max n-1 grana, te bi ukupan broj grana koje polazi iz v1 i v2 bio manji ili jednak od 2(n-1), a po polaznoj pretpostavci važi da $d(v1) + d(v2) \geq 2n$ (KONTRADIKACIJA, sve grane iz čvorova v1, v2 vode ka nekim čvorovima uparivanja, inače bi bilo moguće proširiti uparivanje).

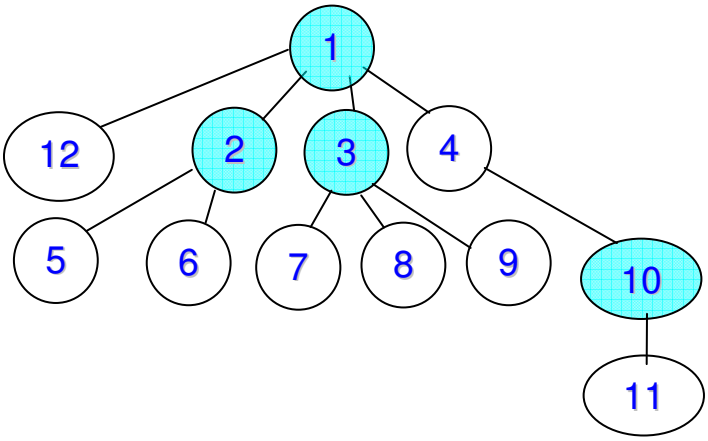
Složenost algoritma je $O(|V|^2)+O(|V|+|E|)$ jer se polazno uparivanje mora povećati za manje od *n* grana

13. Pokrivač grana neusmerenog grafa G = (V,E) je skup čvorova U takav da je svaka grana iz E susedna bar jednom čvoru iz U. Konstruisati algoritam linearne složenosti za nalaženje najmanjeg pokrivača grana datog stabla.



Rešenje:
Crveni čvorovi na slici su pokrivači grana u svakom grafu. Da li su najmanji pokrivači grana svog grafa?

Najmanji pokrivač grana za dato stablo = skup čvorova {1, 2, 3, 10}



Posmatrajmo proizvoljni list v *(v=11)* i njegovog oca w (*w=10*). Grana (v,w) može se pokriti čvorom v, ali je bolje pokriti čvorom w, iz razloga što čvor v pokriva samo (v,w), a čvor w pokriva i neke druge grane. Preciznije, postoji **minimalni** pokrivač grana koji sadrži w.
Zato, ako izaberemo čvor w, uklonimo ga sa svim njemu susednim granama i indukcijom rešimo preostali problem, dolazimo do minimalnog pokrivača.
Vremenska složenost algoritma je proporcionalna broju čvorova u delu gde tražimo listove i broju grana u delu gde uklanjamo naslednike oca, tj. linearna je.

14. Konstruisati algoritam koji za dati graf G = (V,E) utvrđuje da li sadrži podskup čvorova U koji je istovremeno i minimalni pokrivač grana i maksimalni nezavisni skup (tj. ne postoji grana između bilo kojih od čvorova).
Rešenje:
Pretpostavimo da u G postoji takav podskup $U \subset V$.

Između čvorova u U ne sme postojati grana (jer je U nezavisan skup), a svaki čvor V\U mora biti povezan sa nekim cvorom iz U. (Inace, U ne bi bio maksimalan nezavisan skup).
Između čvorova u V\U ne sme postojati grana (jer je U pokrivač grana), a svaki cvor iz U mora biti povezana sa nekim cvorom iz V\U (jer je U minimalni pokrivac grana).
Dakle, graf je bipartitivni bez izolovanih cvorova.

Obrnuto, ako je G=(U_1,U_2,E) bez izolovanih cvorova, onda se bilo U_1, bilo U_2 moze uzeti za trazeni skup U.
Provera da li je graf bipartitan se vrši u linearnom vremenu (zad 6.22 udžbenika), a takođe i provera da li postoje izolovani čvorovi.

Transportne mreže (mreža fluida, naftovod, telefonska mreža, distributivna mreža proizvoda, vojnici u ratu, mreža pruga ili drumskih puteva,...)

Teorema o uvećavajućem putu transportne mreže i algoritam za rešavanje transportnog problema
Transportna mreža se može definisati na sledeći način:
Neka je G = (V, E) usmereni graf sa dva posebno izdvojena čvora, **s(izvor) sa ulaznim stepenom 0** i **t(ponor) sa izlaznim stepenom 0**. Svakoj grani e koja pripada E pridružena je pozitivna težina s(e), kapacitet grane e. Kapacitet grane toka je mera toka koji može biti propušten kroz granu. Za ovakav graf kažemo da je mreža.
Tok je funkcija f definisana na E koja zadovoljava sledeće uslove:

1. tok kroz proizvoljnu granu ne može da premaši njen kapacite
2. ukupan tok koji ulazi u neki čvor je jednak toku koji izlazi iz tog čvora (nestišljivost, zakon očuvanja)

Povećavajući put u odnosu na zadati tok f je usmereni put od s do t koji se sastoji od grana iz G, ne obavezno u istom smeru; svaka od tih grana (v, u) treba da zadovolji tačno jedan od sledećih uslova:

1. (v, u) ima isti smer kao i u G, i f(v, u) < c(v, u). U tom slučaju grana (v, u) je direktna (forward) grana. Direktna grana ima kapacitet veći od toka,pa se može povećati tok kroz u. Razlika c(v, u) i f(v, u) zove se slek te grane.

2. (v, u) ima suprotan smer u G, i f(v, u) > 0. U ovom slučaju grana (v, u) je povratna (backward) grana. Deo toka iz povratne grane se može pozajmiti.

Povećavajući put je uopštenje alternirajućeg puta i ima isti smisao za transportne mreže kao alternirajući put za bipartitivno uparivanje.
Teorema: Tok kroz mrežu je optimalan akko u odnosu na njega ne postoji povećavajući put.
Teorema o povećavajućem putu neposredno se transformiše u algoritam. Polazi se od toka 0, traže se povećavajući putevi, i na osnovu njih se povaćava tok, sve do trenutka kada povećavajući putevi ne postoje. Traženje povećavajućih puteva se može izvesti na sledeći način. Definišemo rezidualni graf u odnosu na mrežu G = (V, E) i tok f, kao mrežu R = (V, F) sa istim čvorovima, istim izvorom i ponorom, ali promenjenim skupom grana i njihovim kapacitetima. Svaku granu e = (v, w) sa tokom f(e) zamenjujemo sa najviše dve grane e' = (v, w) (ako je f(e) < c(e)), kapacitet e' jednak je sleku grane e: c(e') = c(e) - f(e), odnosno e'' = (w, v) (ako je f(e) > 0, kapacitet e'' je c(e'') = f(e)). Ako se na ovaj način dobiju dve paralelne grane, zamenjuju se jednom, sa kapacitetom jednokom zbiru kapaciteta paralelnih grana. Grane rezidulanog grafa odgovaraju mogućim granama povećavajućeg puta. Njihovi kapaciteti odgovaraju mogućem povećanju toka kroz te grane. Prema tome, povećavajući put je običan usmereni put od s do t u rezidualnom grafu. Konstrukcija rezidualnog grafa zahteva O(|E|) koraka.
Edmonds i Karp su pokazali da je među povećavajućim putevima uvek bolje birati one sa manjim brojem grana. Tako je algoritam u najgorem slučaju polinomijalan u odnosu na veličinu ulaza.

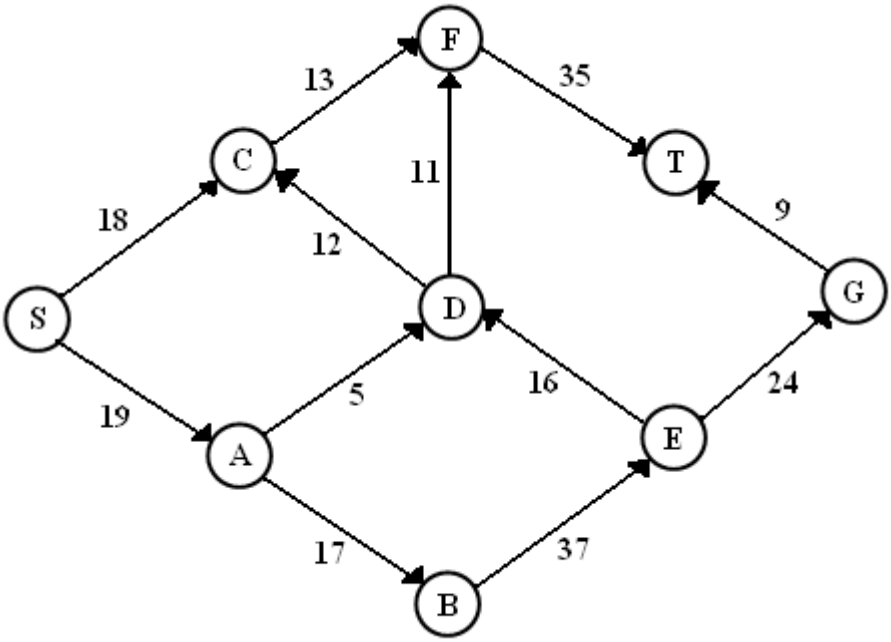
Maksimalni protok grafa se koristi u problemima kada se iz jednog mesta u drugo tj. od izvora do prijemnika treba premostiti neka količina preko veza određene nosivosti. Na primer u vodoinstalacijama, ako imamo izvor vode S(kao početni čvor grafa) i prijemnik vode T(kao krajnji čvor grafa), ostali čvorovi grafa predstavljaju raskršća, a usmerene grane su cevi koje povezuju čvorove tako da voda ide samo u jednom smeru i te cevi imaju svoj maksimalni protok.

Zadatak se rešava tako sto se pretragom po širini ili dubini određuju mogući putevi od izvora S do prijemnika T. Nakon svakog pronađenog mogućeg puta, traži se minimalni protok kroz sve cevi tog puta tj. trazi se cev sa najmanjom nosivošću, jer je nemoguće da prođe više vode kroz cev nego što ona može da propusti. Kroz sve cevi tog trenutnog puta se propusti taj minimum i onda se nastavlja pretraga po širini ili

dubini do nalaženja novog puta gde se opet minimum traži i tako sve iznova. Voda(količina) koja izađe iz izvora u svakom koraku mora da bude jednaka onoj količini koja dolazi do prijemnika.

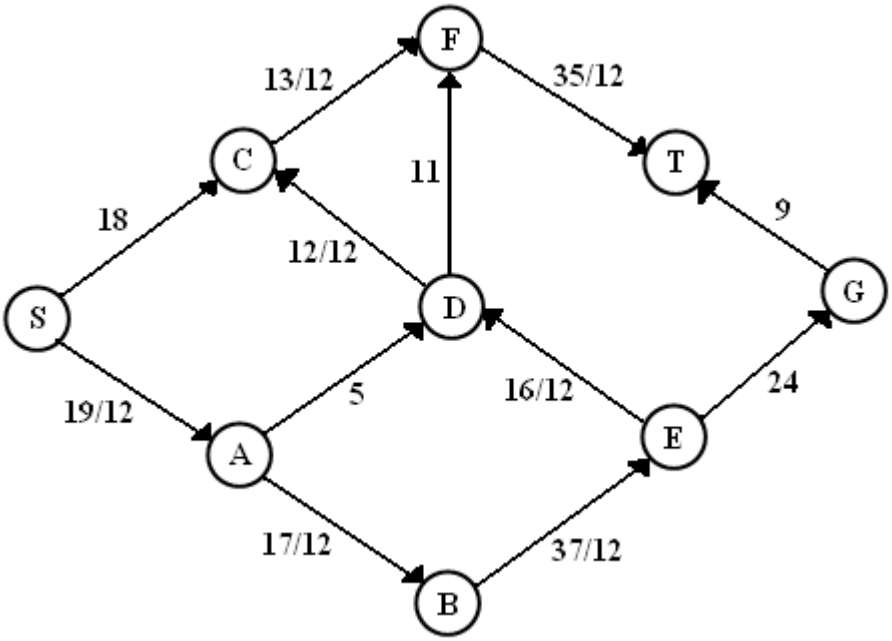
Zbog rešavanja zadatka uz pomoć rezidualnog grafa, za svaku cev postavlja se imaginarna u kojoj voda prolazi u suprotnom smeru i količina vode koja prolazi kroz imaginarnu cev je samo suprotna po znaku.

1. Pronađite maksimalni protok datog grafa.

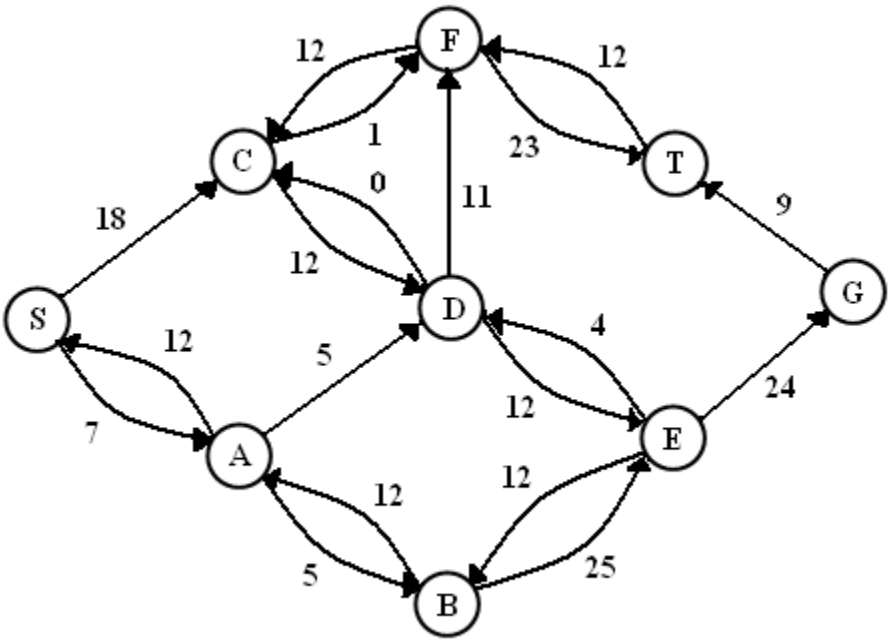


Prvi pronađeni put je SABEDCFT(pretraga po dubini)
Nosivosti grana tog puta su: 19,17,37,16,12,13,35

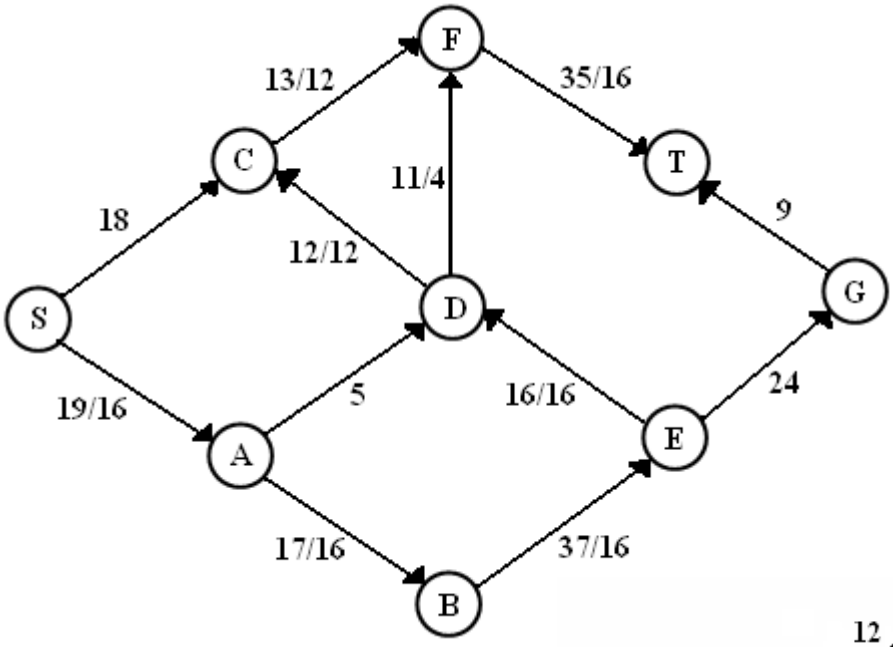
Minimum je 12 i ta se količina prva propušta kroz grane tog puta:



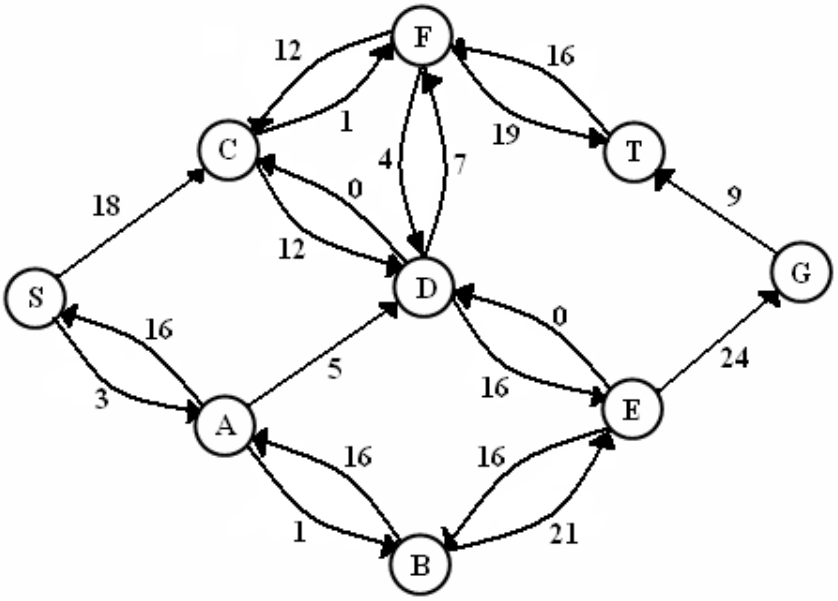
Sa imaginarnim cevima:



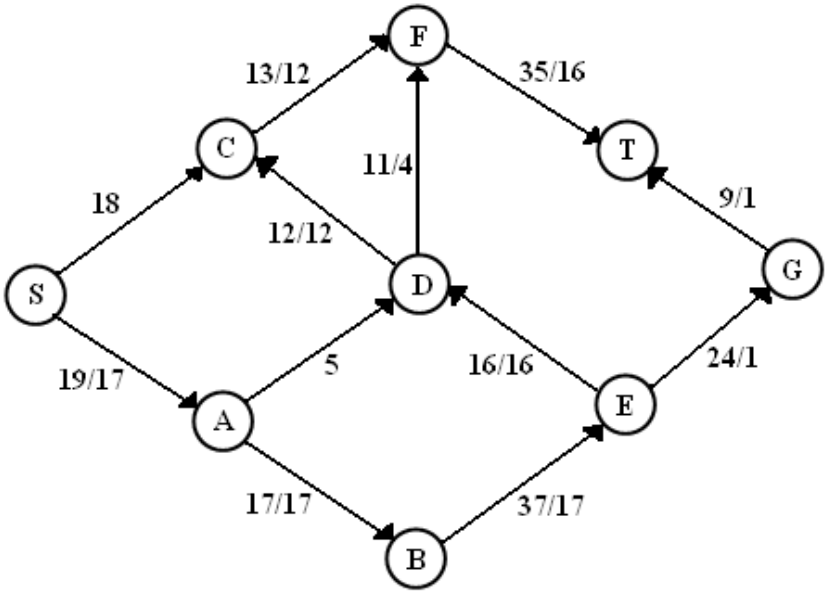
Drugi pronađeni put je SABEDFT
Nosivosti grana tog puta su: 7,5,25,4,11,23
Minimum je 4 i ta kolicina se sledeca propusta:

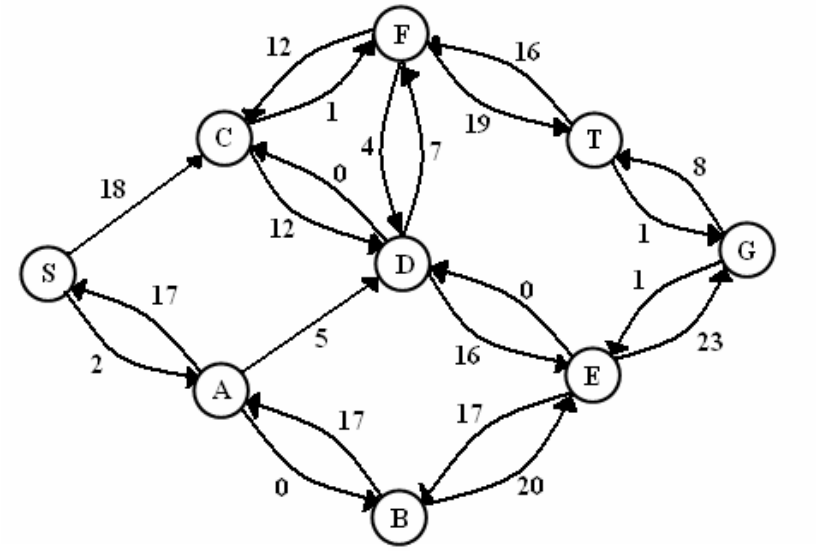


Sa imaginarnim cevima:

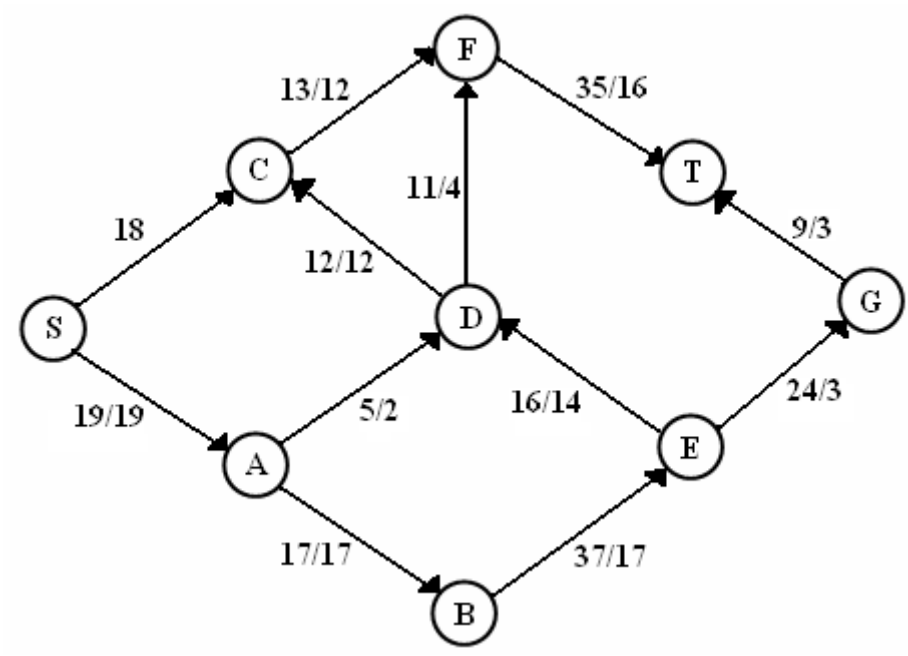


Treci pronadjeni put je SABEGT
Nosivosti grana tog puta su: 3,1,21,24,9
Minimum je 1 i ta se kolicina sledeca propusta:



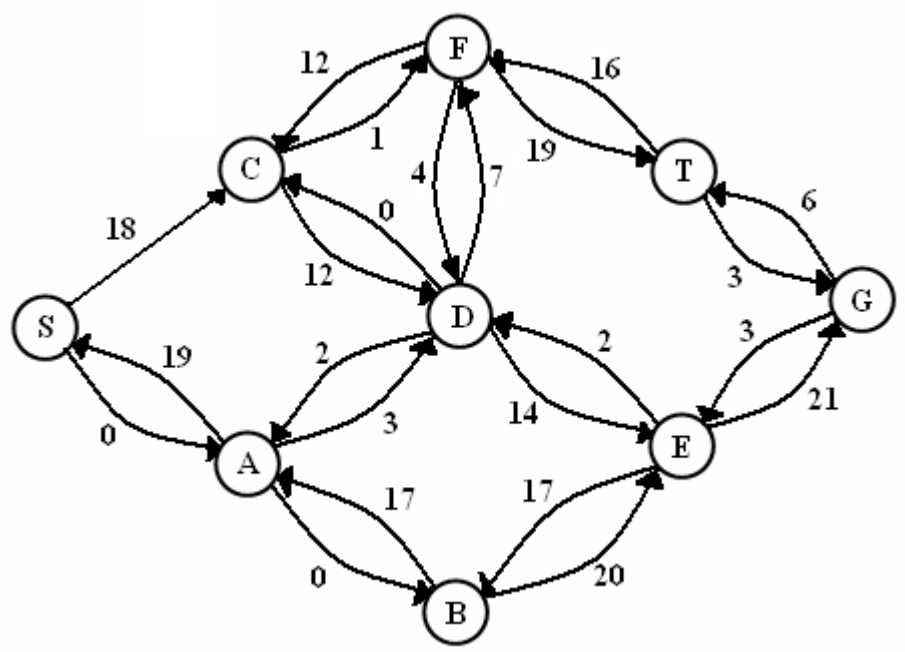


Sa imaginarnim cevima:



Četvrti pronađeni put je SADEGT
Nosivosti grana tog puta su: 2,5,16,23,8
Minimum je 2 i ta se kolicina sledeca propusta:

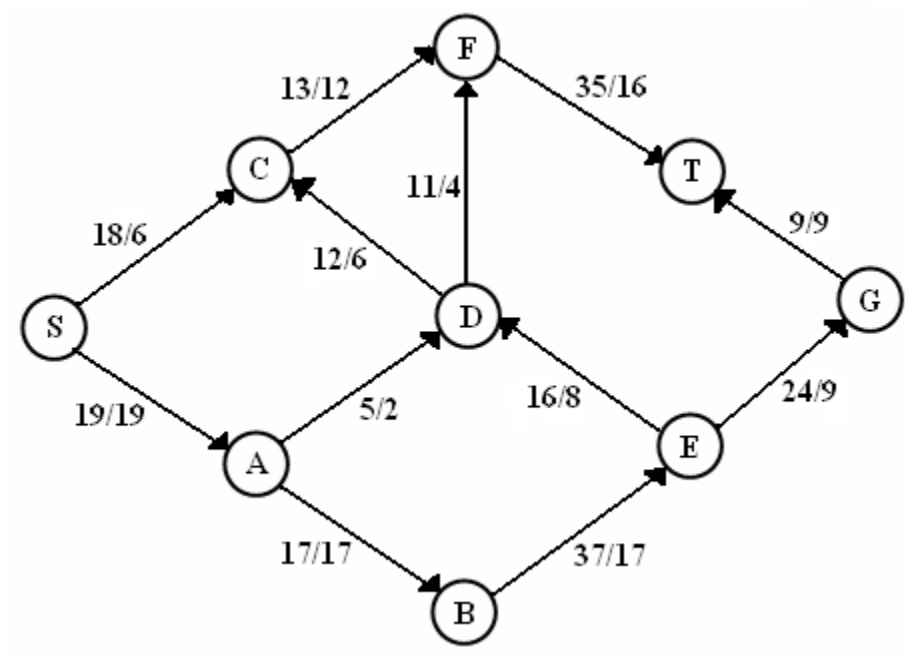
Sa imaginarnim cevima:



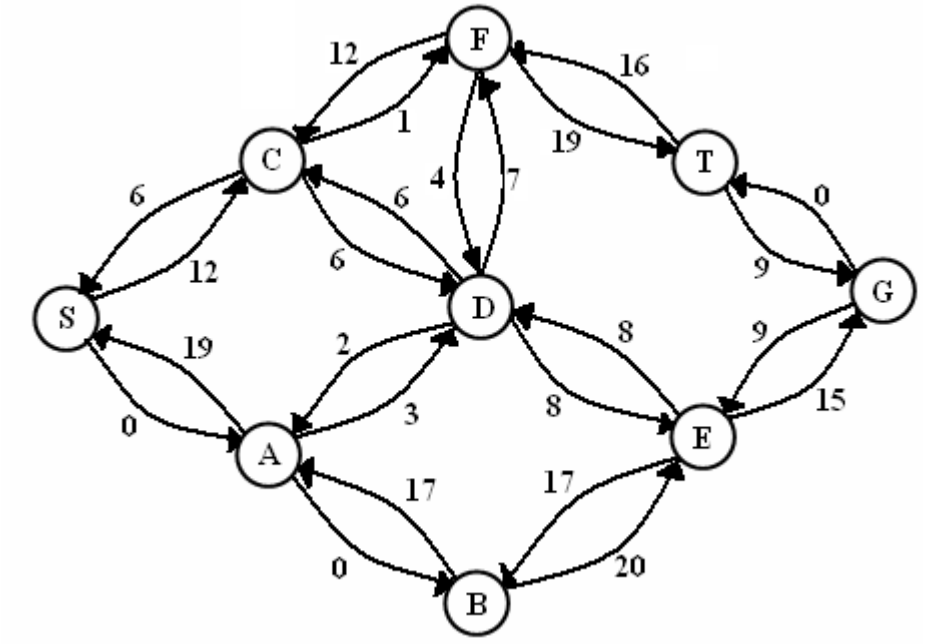
Peti pronadjeni put je SCDEGT
Nosivosti grana tog puta su: 18,12,14,21,6

Minimum je 6 i ta se kolicina sledeca propusta:

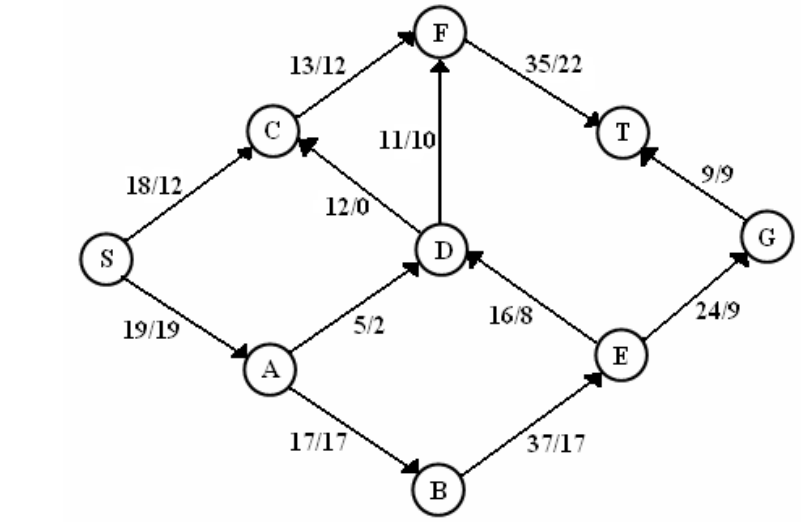
Sa



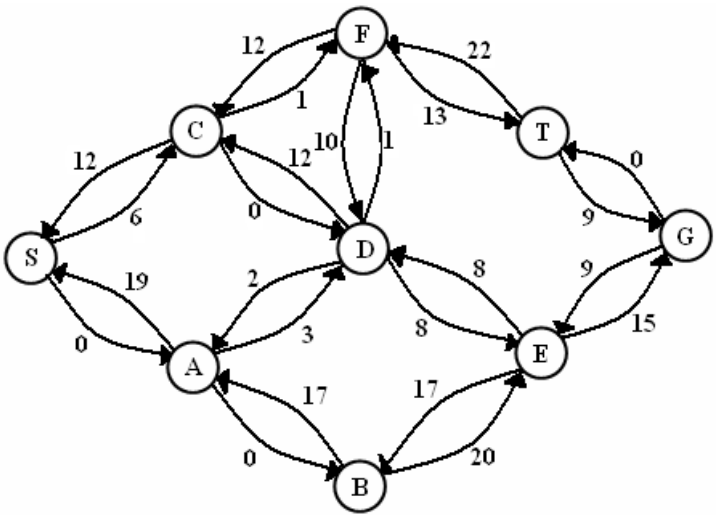
imaginarnim cevima:



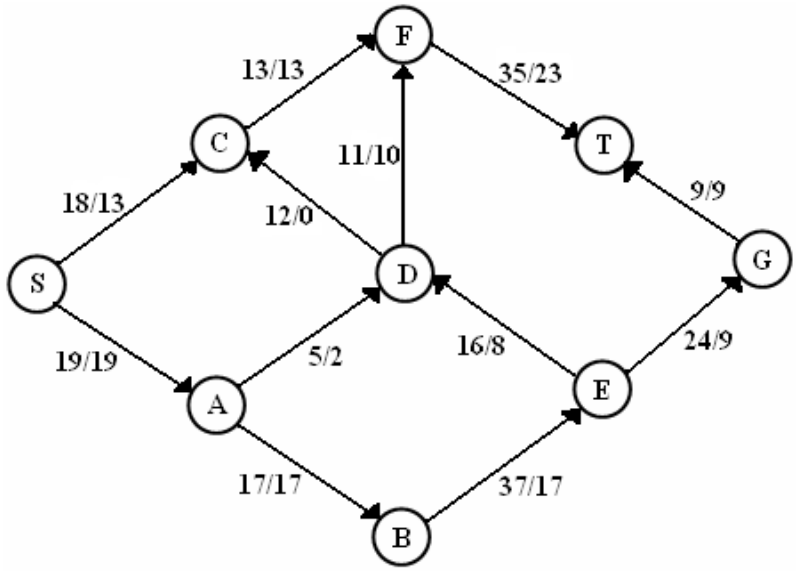
Šesti pronadjeni put je SCDFT
Nosivosti grana tog puta su: 12,6,7,16
Minimum je 6 i ta se kolicina sledeca propusta:



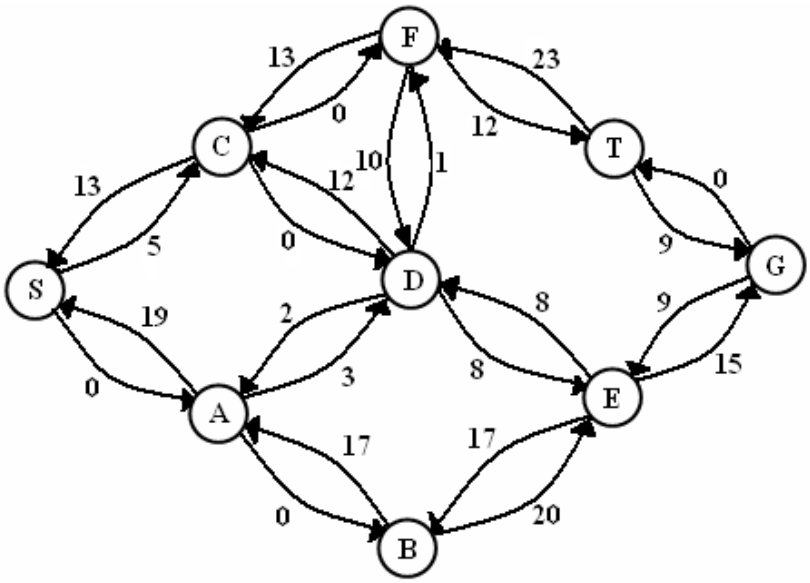
Sa imaginarnim cevima:



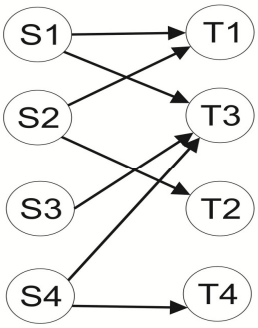
Sedmi pronadjeni put je SCFT
Nosivosti grana tog puta su: 6,1,13. Minimum je 1 i ta se kolicina sledeca propusta:

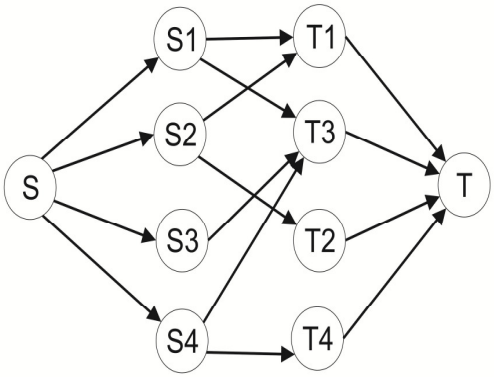


Sa imaginarnim cevima:

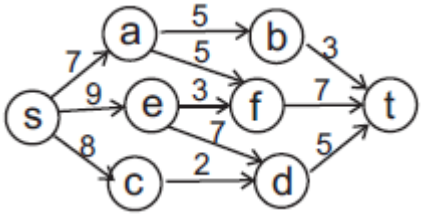


Maksimalni protok u ovom primeru je 32 i predstavlja zbir minimuma pri svakom novom određivanju mogućeg puta.
12+4+1+2+6+6+1=32
Kada graf ima vise izvora i(ili) vise prijemnika zadatak se resava tako sto se postavi novi izvor iz kog se dolazi direktno u sve izvore i(ili) novi prijemnik u koji se dolazi direktno iz svih prijemnika.

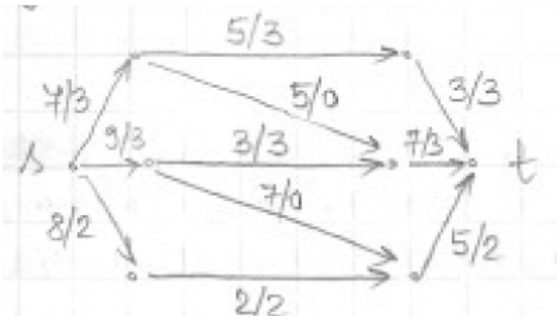




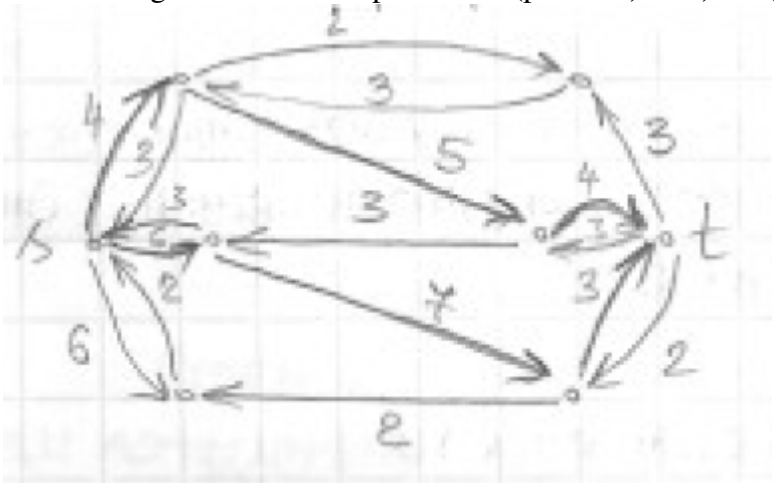
2. (primer iz udžbenika) Dat je usmeren graf G = (V,E) sa dva izdvojena čvora *s* i *t*. Granama grafa pridruženi su brojevi – njihovi kapaciteti. Odrediti optimalni tok kroz mrežu.



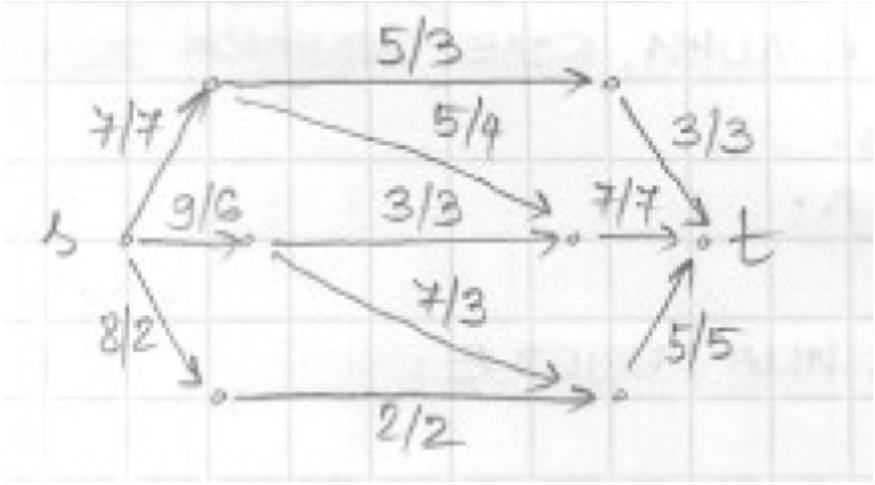
Resenje:
Nađemo neki tok. Svaka grana je oznacena sa a/b gde a =kapacitet, b=trenutni tok.
Možemo da krenemo puštanjem protoka 3 kroz grane kapaciteta 7,5,3 (put **sabt)**
Možemo da krenemo puštanjem protoka 3 kroz grane kapaciteta 9,3,7 (put **seft)**
Možemo da krenemo puštanjem protoka 2 kroz grane kapaciteta 8,2,5 (put **scdt)**



Rezidualni graf u odnosu na pušten tok (put **sabt, seft, scdt)** :



U rezidualnom grafu se mogu izdvojiti 2 nezavisna(disjunktna) povećavajuća puta
1. put **saft** kapaciteta min{4,5,4}=4
2. put **sedt** kapaciteta min {6,7,3}=3
Uzimajući ovo u obzir (tj. pustimo protok 4 na putu **saft** i protok 3 na putu **sedt**)dobija se sledeći protok:



Ovaj tok je optimalan, jer su tokovi kroz grane od b, f, d ka t (koji čine presek određen skupom V\{t}) jednaki njihovim kapacitetima (tok/kapcitet za granu **bt** je 3/3, za granu **ft** je 7/7, za granu **dt** je 5/5).
Slozenost: $O(|V|^3)$

3. Transportna mreža može imati više izvora i ponora, umesto po jednog. Rešiti problem maksimizacije toka u ovom slučaju.

4. Profesor Krstić ima dvoje dece koja na žalost ne vole jedno drugo. Problem je toliko ozbiljan da ne samo da odbijaju da zajedno idu do škole već ne žele da prođu nijednim delom puta kojim je drugo dete toga dana prošlo. Ipak, ne smeta im da im se putevi ukrštaju na uglu. Srećom i profesorova kuća i škola su na uglu, ali on nije siguran da li je moguće da pošalje oba deteta u istu školu. Profesor ima mapu svoga grada. Pokazati kako se ovaj problem može formulisati u terminima maksimizacije toka.

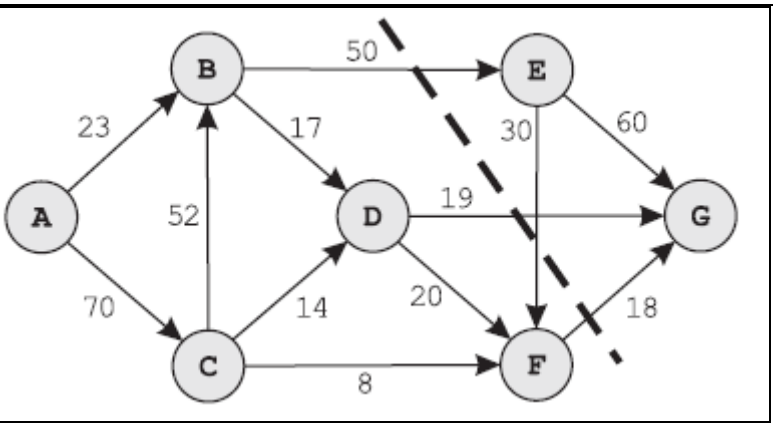
Rešenje:
Formulacija:
Čvor – ugao
Grane – trotoari i pešački prelazi koji povezuju uglove.
Polazni čvor s – kuća profesora Krstića
Izlazni čvor t – škola
Sve grane imaju težninu 1, jer najviše jedno dete može hodati duž grana grafa (trotoar, pešački prelaz)

Potrebno je rešiti problem maksimalnog protoka od s do t.
Ako je mrežni protok barem 2, to znači da postoji dovoljno pešačkih kapaciteta unutar grada tako da oba deteta mogu da pešače od s do t duž jedinstvenog puta (grana) sa mogućim ukrštanjima na uglu (čvor grafa).
U suprotnom, profesor Krstić mora upisati decu u 2 različite škole.

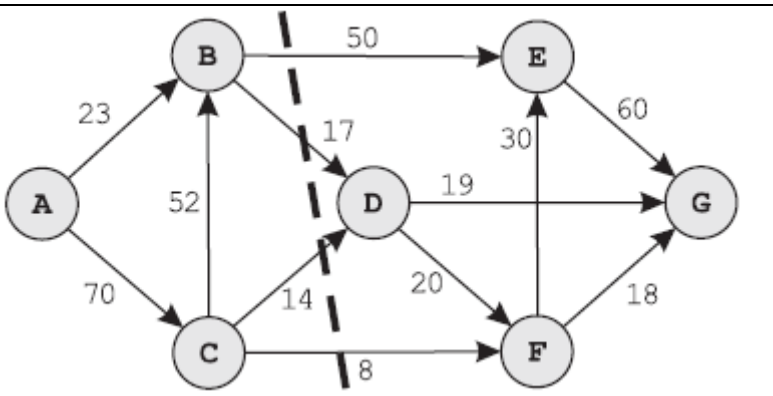
5.
Maksimalan tok kroz transportnu mrežu == minimalna suma težine grana čijim uklanjenjem se ne može doći od izvor s do ponora t

Teorema 1: Min-cut teorema tvrdi da je maksimalan tok kroz mrežu jednak minimalnom kapacitetu koji se mora ukloniti iz mreže kako ne bi bilo protoka od izvora do ponora.

To jest: Minimalna suma težine grana grafa, koje je potrebno ukloniti iz grafa da se ne bi moglo doći od izvora do ponora, jednaka je problemu *Network flow*. Znači:
Minimum cut == Network flow



Na slici protoka kroz grafa, isprekidana linija označava *Minimum cut*. U tom slučaju će *Minimum cut* preseći grane BE, DG i FG (grane su usmerene s leve strane isprekidane linije prema desnoj, zato ne brojimo granu EF). Tada se više ne može od čvora A doći do čvora G.
Uočite da je suma težina grana BE, DG i FG jednaka 50+19+18 =87, a toliko iznosi i *Network flow* (od čvora A do čvora G) za prikazan graf.



Ako preusmerimo granu EF, onda je *Network flow* za dati graf 89, kao i *Minimum cut*.
Ta situacija prikazana je na grafu sa slike levo.

Minimum cut postižemo presecanjem grana BE, BD, CD i CF. Suma težina tih grana je 50+17+14+8 = 89.

Vrednost za *Minimum cut* može se jednostavno izračunati najvećim tokom kroz graf, ali utvrditi tačno o kojim se granama radi može biti vrlo teško. Jedan način da se odrede grane jeste pokušaj da se izbace sve kombinacije grana koje Ford–Fulkersonov algoritam postavi na 0 i „poplaviti" (engl. *flood fill*) izvor kako bi se proverilo je li graf i dalje povezan.

Drugi način za utvrđivanja koje grane čine *Minimum cut* jest pokušati izbaciti svaku granu (od lakših prema težim) i ako se *Network flow* smanji za težinu izbačene grane, znamo da je ta grana deo *Minimum cut-a* i valja nastaviti bez nje.

Zadatak:
Unutar računarske laboratorije Matematičkog fakulteta, računari su povezani u mrežu mrežnim kablovima. Između nekih parova računara nalazi se dvosmerni mrežni kabel. Ako eliminišemo neki mrežni kabl pojavljuje se određen gubitak u funkcionisanju. Računar sa oznakom X je veoma važan i do njega ne sme dospeti virus koji se nalazi u računarima sa oznakom Y1, Y2, ..., Yn. Koliki je minimalni gubitak u funkcionisanju koji je potrebno pretrpeti kako bi se zaštitio računar X od virusa?

Skica rešenja:
Mreža računara predstavlja graf nad kojim moramo napraviti *Minimum cut* od računara Y1, Y2, ..., Yn do X.

Računari Y1, Y2, ..., Yn su izvori, a X je ponor. Budući da ima više izvora, moramo dodati virtuelan izvor s granama beskonačne težine do izvora Y1, Y2, ..., Yn.
Kako je polazni graf neusmeren, onda svaka grana od čvora A do čvora B težine C se razmatra u dva pojavna oblika, tj. dve usmerene grane, od A do B i od B do A, obe težine C. Nad grafom je potrebno primeniti Ford-Fulkersonov algoritam za nalaženje najvećeg toka kroz graf i ispišemo rešenje.

6. Povezanost neusmerenog grafa G = (V,E) je minimalni broj *k* grana koje se moraju ukloniti da bi graf postao nepovezan. Na primer, povezanost je jednaka 1 za stablo, a 2 za ciklus. Pokazati kako je moguće ovaj problem rešiti pokretanjem algoritma za određivanje maksimalnog toka na maksimalno |V | različitih mreža, pri čemu svaka od njih ima O(|V |) čvorova i O(|E|) grana.

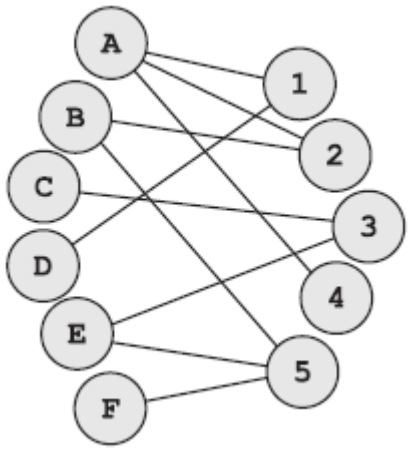
Rešenje:

Želimo da izračunamo povezanost neusmerenog grafa G = (V; E) pokretanjem algoritma za određivanje maksimalnog protoka na maksimalno |V | različitih mreža iste veličine kao i G.
Označimo sa Guv usmerenu verziju grafa. Takav graf Guv ćemo smtrati transportnom mrežom gde s = u, t = v.
Postavimo kapacitet svake grane na 1, tako da broj grana koje prelaze neki razrez grafa(engl. cut) je jednak kapacitetu razreza. Neka je f_{uv} maksimalno protok kroz Guv.
Poveznaost grana se moze izracunati nalazenjem min $|f_{ux}|$, $x \in V\{u\}$. Ovo se moze uociti koristeci **max-flow min-cut teoremu (Ford-Fulkerson).**

7. Neka je G = (V,E) transportna mreža sa izvorom *s*, ponorom *t* i celobrojnim kapacitetima. Pretpostavimo da nam je poznat maksimalni tok u G.
(a) Pretpostavimo da je kapacitet jedne grane $(u, v) \in E$ povećan za 1. Dati algoritam složenosti O(|V |+|E|) za ažuriranje maksimalnog toka.
(b) Pretpostavimo da je kapacitet jedne grane $(u, v) \in E$ smanjen za 1. Dati algoritam složenosti O(|V |+|E|) za ažuriranje maksimalnog toka.

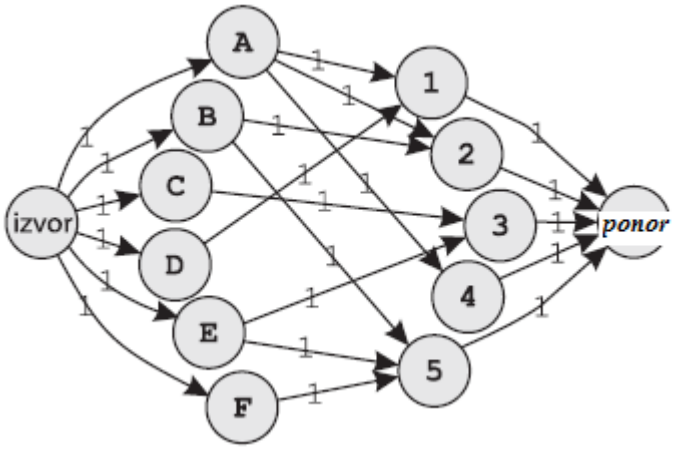
8. Bipartitni graf G = (V,E), gde je $V = L \cup R$, je d-regularan ako svaki čvor $v \in V$ ima stepen tačno d.
(a) Pokazati da za svaki d-regularni bipartitni graf važi: |L| = |R|.
(b) Svesti problem određivanja maksimalnog d-regularnog bipartitnog uparivanja na problem određivanja maksimalnog toka u mreži. Pokazati da je maksimalna vrednost toka u toj formulaciji |L|.
(c) Dokazati da svaki d-regularni bipartitni graf ima uparivanje kardinalnosti |L|.

9.
Optimalno ili najveće moguće uparivanje u bipartitivnom grafu (Maximum bipartite matching)



Pogledajmo graf sa slike i čvorovi nazvane po slovima, kao i čvorovi obeležene ciframa. Graf je bipartitivan jer su sve relacije između slova i cifri.
Koliko se najviše parova može upariti ako svako slovo može biti uparen sa samo jednom cifrom, a svaka cifra sa samo jednim slovom?

Navedeni problem može se rešiti pomoću *Network flowa* (slika niže).



Sve grane usmerimo od slova prema brojkama i damo im protočnost jednaku 1.

Dodamo čvor koji predstavlja izvor i od njega do svakog čvora nazvanog slovom spojimo granu čija je protočnost jednaka 1. Time omogućujemo slovima da koriste granu samo prema jednoj cifri.

Sve čvorove koji su označeni cifrom povežemo prema novododanom čvoru koji predstavlja ponor i damo im protočnost jednaku 1. Time omogućimo svakoj cifri da se poveže samo s jednim slovom.

Nakon što napravimo *Network flow*, kao rezultat dobićemo najveći broj parova koji se mogu upariti, a gledajući koje grane su postavljeni na 0 možemo saznati ko je s kim uparen.

Šta u stvari radi algoritam *Network flow* radi?

Recimo da prvo pronađe put Izvor-A-1-Ponor, pa zato postavi protočnost grane od A1 na 0, a grane 1A na 1. To je isto kao da smo jednostavno granu A1 okrenuli tako da imamo granu 1A.

Sledeći put se pronađe: na primer, put Izvor-B-2-Ponor, pa se grana B2 pretvori u 2B.
Sledeći put koji se pronađe je, na primer, Izvor-C-3-Ponor, pa je sada grana C3 postala 3C.

Sledeći put je npr. Izvor-D-1-A-2-B-5-Ponor. U tom putu koristimo granu 1A (nastao okretanjem grane A1) i granu 2B (nastao okretanjem B2). Grana D1 se pretvara u 1D (to znači da je D uparen s 1). Grana 1A se vraća u A1 (znači A i 1 više nisu upareni). Grana A2 postaje 2A (znači da je A uparen s 2). Grana 2B postaje B2 (znači da B više nije uparen s 2). Grana B5 postaje 5B (znači B je uparen s 5).
Zapravo smo u ovoj iteraciji *Network flowa* preusmerili A s 1 na 2, B s 2 na 5, a D povezali s 1.

Sledeći put koji Ford-Fulkersonov algoritam pronađe je npr. Izvor-E-5-Ponor, pa se grana E5 pretvori u 5E (znači da je E uparen s 5). Sada više nema puteva i Ford-Fulkersonov algoritam završava. Ukupan *Network flow* je 5, što znači da se najviše 5 parova može upariti. Umesto zadnjeg navedenog puta Ford-Fulkersonov algoritam je mogao pronaći i Izvor-F-5-Ponor.

Različiti putovi bi različito povezali parove, ali bi broj parova i dalje bio 5.

Primetite da se ponašanje Ford-Fulkersonovog algoritma za problem najvećeg mogućeg uparivanja na bipartitivnom grafu može simulirati i rekurzijom, te brže pronaći.

10. *Unosi se n1 (broj čvorova iz prvog skupa), n2 (broj čvorova iz drugog skupa) i m, zatim m grana opisanih s po dva broja a i b. Čvor a (0 <= a < n1) iz prvog skupa je povezan granom s čvorom b (0 <= b < n2) iz drugog skupa. Ispiši koliko najviše parova čvorova se može povezati (nekim od postojećih grana) tako da je svaki čvor povezan s najviše jednim čvorom.*

Primer unosa:
6 5 10
0 0 0 1 0 3 1 1 1 4
2 2 3 0 4 2 4 4 5 4

Unos odgovara gore navedenom grafu slova i cifri.
Odgovarajući ispis:
5
Sledi dodatni ispis:
A + 4
B + 2
C + 3
D + 1
E + 5

Rešenje:

```
01. #include <vector>
02. #include <iostream>
03. using namespace std;
04.
05. vector<vector<int> > graf;
```

```
06. vector<int> spojenSa;
07. vector<int> bio;
08. int n1,n2,m;
09.
10. bool DFS(int x) {
11. bio[x]=1; // kako ne bi pozvali rekurziju za isti cvor
12. int ko;
13. for (int i=0;i<graf[x].size();i++) {
14.     ko=graf[x][i];
15.  if (spojenSa[ko]==-1 || (!bio[spojenSa[ko]]
16.  && DFS(spojenSa[ko]))) {
17.          spojenSa[ko]=x;
18.           return 1;
19.    }
20. }
21. return 0;
22. }
23.
24. int main() {
25. cin >> n1 >> n2 >> m;
26. vector<int> vi;
27. graf.insert(graf.begin(),n1,vi);
28. spojenSa.insert(spojenSa.begin(),n2,-1); // -1 znaci slobodan
29. int a,b;
30. for (int i=0;i<m;i++) {
31. cin >> a >> b; graf[a].push_back(b); }
32.
33. int resenje=0;
34. for (int i=0;i<n1;i++) {
35.   bio.clear();
36.    bio.insert(bio.begin(),n1,0);
37.    resenje+=DFS(i);
38. }
39. cout << resenje << endl;
40. cout << " Sledi dodatni ispis: " << endl;
41. for (int i=0;i<n2;i++)
42.     cout << (char)(i+'A') << " + " << spojenSa[i]+1 << endl;
43. return 0;
44. }
45.
```

Objašnjenje:
Od 25. do 31. linije je unos grafa.
U vektor graf zapisujemo spisak susednih čvorova.
Vektor spojenSa je dimenzije n2 i beleži -1 ako niko nije spojen s odgovarajućim čvorom, dok inače beleži broj čvora.

U 33. liniji inicijalizujemo rešenje na 0.
U 34. liniji prolazimo kroz sve čvorove iz skupa veličine n1 i svaki uparujemo (37. linija).
DFS vraća 1 ako se čvor uspe spojiti, a inače 0.

Pre poziva DFS prvo postavimo u 35. i 36. liniji sve elemente vektora bio (veličine n1) na 0, kako bi rekurzija mogla pamtiti za koje čvorove je pozvana i ne pozivati se ponovno za isti čvor. DFS prima čvor iz skupa veličine n1 za koji se poziva.

```
10. bool DFS(int x) {
11. bio[x]=1; // kako ne bi pozvali rekurziju za isti cvor
12. int ko;
13. for (int i=0;i<graf[x].size();i++) {
14.     ko=graf[x][i];
15.  if (spojenSa[ko]==-1 || (!bio[spojenSa[ko]]
16.  && DFS(spojenSa[ko]))) {
17.          spojenSa[ko]=x;
18.           return 1;
19.    }
20. }
21. return 0;
22. }
```

U 11. liniji markiramo da smo već posetili čvor x.
U 14. liniji promenljiva ***ko*** je neki čvor iz skupa veličine n2 dohvatljiv granom.

15. i 16. linija koda su složenije i zapravo odgovaraju na pitanje može li se x spojiti s promenljivom ko. Ako se može, onda u 17. liniji beležimo s kime je ko spojen i vraćamo 1.

U 15. liniji prvo proveravamo da li je *ko* slobodan: spojenSa[ko]==-1.
Ako je to tačno, onda se ostatak if naredbe ne izvršava.
Ako to nije istinito, treba proveriti može li se onaj s kime je *ko* spojen prespojiti na neki drugi čvor. To radimo samo ako rekurzija nije već pozvana za čvor s kime je *ko* spojen.
DFS će pokušati čvor spojenSa[ko] prespojiti na neki drugi čvor različit od ko, jer je spojenSa[ko]==-1 neistinit, kao i !bio[spojenSa[tko]].
Navedeni rekurzivni poziv je nešto složeniji, te zavređuje dodatnu ponovljenu analizu.

Složenost navedene implementacije je u najgorem slučaju O(V*E), tačnije O(n1*E).
Rekurziju pozivamo n1 puta, a rekurzija pamti u kojim čvorovima je bila, tako da u najgorem slučaju prođe preko svih E grana.

11. Potrebno je uneti brojeve u n × n matricu sa celobrojnim vrednostima između 0 i granice *k*, tako da suma svih brojeva u svakoj vrsti, odnosno koloni, bude jednaka jednom od *2n* brojeva datih unapred. Na primer, sledeća instanca:

$$\begin{array}{c} \\ 6 \\ 9 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccc} 17 & 5 & 4 \end{array} \\ \begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} \end{array}$$

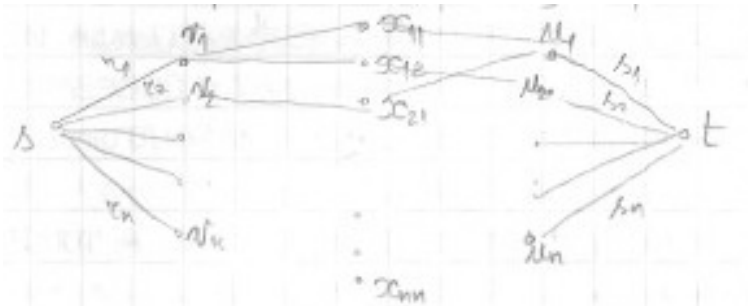
za k=9 ima rešenje:

$$\begin{array}{c} \\ 6 \\ 9 \\ 11 \end{array} \begin{array}{c} \begin{array}{ccc} 17 & 5 & 4 \end{array} \\ \begin{pmatrix} 6 & 0 & 0 \\ 2 & 3 & 4 \\ 9 & 2 & 0 \end{pmatrix} \end{array}$$

Formulisati i rešiti ovaj problem kao problem maksimizovanja toka.

Resenje:
Posmatramo n*n matricu sa zbirovima c_i po kolonama i r_i po vrstama i nepoznatim elementima m*ij.* Mrezu definisemo na sledeci nacin:
izvor s ima grane ka n cvorova v_i koji su kapaciteta r_i
ponor t ima grane od n cvorova u_j koji su kapaciteta c_i

Zatim definišimo n*n cvorova xij sa granama (vi, xij), (xij, uj), sve kapaciteta k. Celobrojni maksimalni tok na ovoj mrezi odredjuje resenje problema sa matricama, gde je tok kroz (xij ,vi) = (xij, uj) =m_{ij}



Dokaz: Posmatrajmo rešenje problema sa matricama. Ono određuje validan tok, jer se nijedan od kapaciteta ne prekoračuje

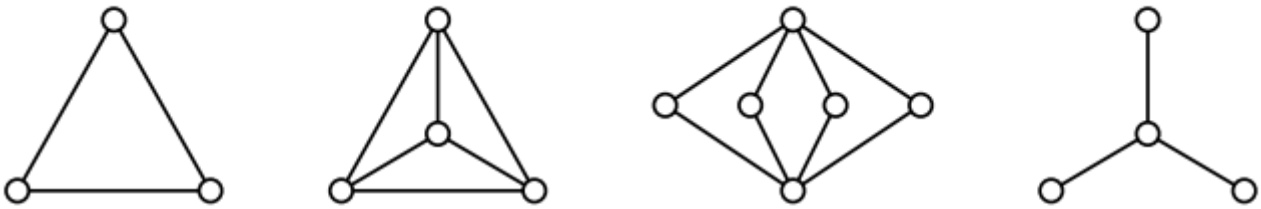
$$\sum_i m_{ij} = \sum_i f(x_{ij}, u_j) = s_j$$

što je kapacitet svih izlaza (a postoji samo 1) iz u_j.
Slično važi i za kolone.
Dalje važi da svi $m_{ij}<k$, i svi tokovi $(v_i,x_{ij})<k$, $(x_{ij,}\, u_j)<k$
Otud i ovi tokovi se ne prekoracuju.

Drugo, postoji presek u mrezi G\{s} sa vrednoscu $\sum_i r_i$

Stoga imamo validan tok i presek iste veličine, te stoga svaki mora biti maksmimalni tok/minimalni presek.

12. Graf koji ima Hamiltonovu konturu je Hamiltonov graf. Hamiltonova kontura sadrži sve čvorove grafa. Identfikujte Hamiltonove grafove na sledećim slikama. Kontura Cn (n>=3) je povezan graf koji ima sve stepene čvorove stepena 2.



C_3 K_4 $K_{2,4}$ S_4

Odgovor: C3, K4

13. Zadat je aciklički usmeren graf *G=(V,E).* Konstruisati algoritam linearne složenosti koji utvrđuje da li u G postoji neki prost put koji sadrži sve čvorove grafa.

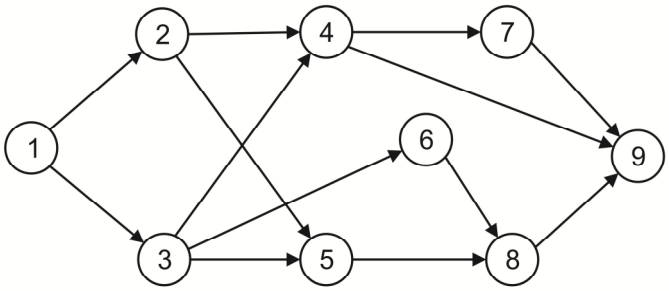
REŠENJE:
Put je prost, ako se svaki čvor pojavljuje u njemu samo jednom.
Hamiltonov put je prosti put u kom se svaki čvor grafa pojavljuje tačno jednom.
U ovom zadataku zahtev je da se konstruiše algoritam za pronalazak Hamiltonovog puta u grafu G.
Ideja je da se konstruiše topološki redosled čvorova grafa G, a potom se proveri da li postoji put tako što se proveri da li susedni čvorovi u topološkom redosledu jesu putno povezani. Ukoliko su povezani, onda Hamiltonov put čine topološki uređeni čvorovi.
U glavi 6.4 knjige je prikazan algoritam za topološko sortiranje acikličkog usmerenog grafa G.

Algoritam topološkog sortiranja je složenosti $O(|E| + |V|)$, tj. linearne složenosti kod grafova.

S druge strane, ako postoji Hamiltonov put, onda topološko sortiranje mora da dovede upravo do onog redosleda čvorova kojim se oni nižu u putu!!!

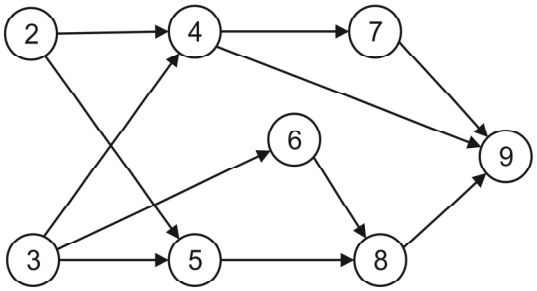
Primer: Algoritam topološkog sortiranja sastoji se u sledećem postupku. Pronađe se čvor bez ulaznih grana, iz sekvencijalnog reda čvorova u skupu V, i zapiše se čvor u niz. Potom se iz grafa ukloni taj čvor i obrišu sve grane koje polaze iz tog čvora. Za novodobijeni graf se ponovo nađe čvor bez ulaznih grana i zapiše u niz, zatim se iz grafa ukloni taj čvor i pobrišu sve grane koje polaze iz tog čvora itd.

Čvor bez ulaznih grana: 1 (postoji po lemi 6.9 u usmerenom acikličkom grafu)



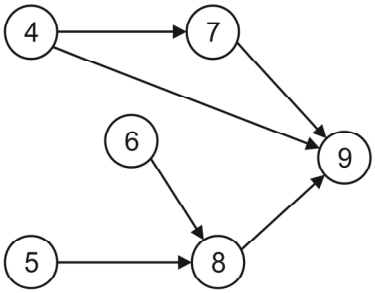
1								

Čvor bez ulaznih grana u novom grafu: 2, 3

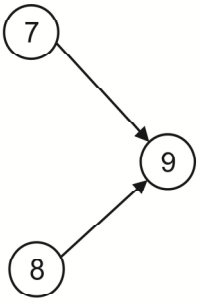


1	2	3						

Čvor bez ulaznih grana u novom grafu: 4, 5, 6



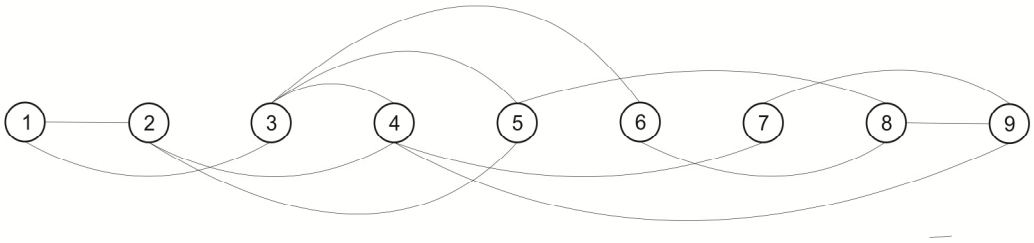
1	2	3	4	5	6			



1	2	3	4	5	6	7	8	

9

1	2	3	4	5	6	7	8	9



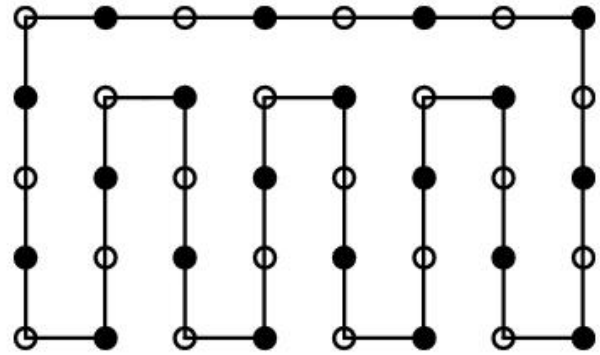
Da li su putno povezani 1,2? Jesu.
Da li su putno povezani 2,3? Nisu.

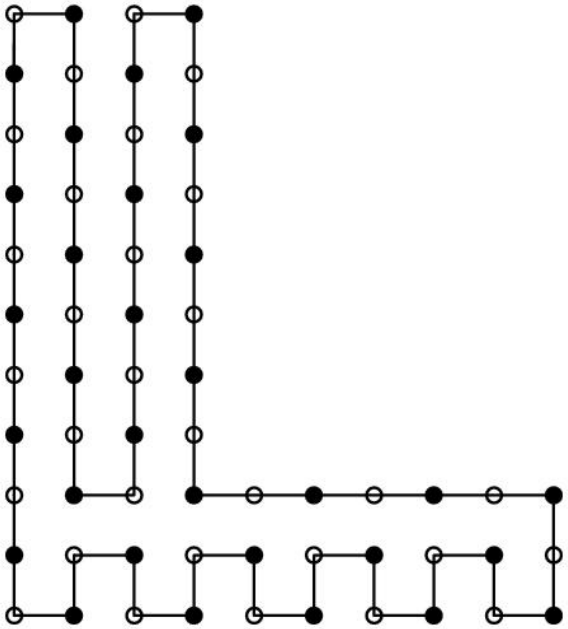
14. Nalaženje Hamiltonovog ciklusa u gustom grafu G = (V,E) (d(v) + d(w) ≥ n za svaki par čvorova (v, w) ∉ E)
Problem: Dat je povezan neusmeren graf G = (V, E), takav da svaki par nesusednih čvorova u i v zadovoljava uslov d(u) + d(v) ≥ n. Pronađi u G Hamiltonov ciklus.
Algoritam se zasniva na indukciji po broju grana koje treba ukloniti iz kompletnog grafa da bi se dobio zadati graf. Baza indukcije je kompletan graf. Svaki kompletan graf sa bar tri čvora sadrži Hamiltonov ciklus, koji je lako pronaći.
Induktivna hipoteza: Umemo da pronađemo Hamiltonov ciklus u grafovima koji zadovoljavaju zadate uslove ako imaju bar $\frac{n(n-1)}{2} - m$ grana.

Sada treba da pokažemo kako pronaći Hamiltonov ciklus u grafu koji ima $\frac{n(n-1)}{2} - (m+1)$ grana i koji zadovoljava uslove zadatka. Neka je G = (V, E) takav graf. Izaberimo dva nesusedna čvora u, v u G, i posmatrajmo graf G' koji se dobija od G dodavanjem grane (u, v). Prema induktivanoj hipotezi mi umemo da pronađemo Hamiltonov ciklus u G' i neka je to $x_1, x_2, \ldots, x_n, x_1$. Ako grana (u, v) nije deo tog ciklusa, onda je taj ciklus deo grafa G, i problem je rešen. U protivnom bez smanjenja opštosti može da se pretpostavi da je v = x_1 и u = x_n. Prema datim uslovima je d(u)+d(v)>=n. Potrebno je pronaći novi Hamiltonov ciklus u grafu.
Tvrdimo da pod zadatim uslovima u G postoje dva čvora x_i , x_{i+1} i grane (v, x_i) , (u, x_{i+1}). Pretpostavimo suprotno, da ni za jedno i, $2 \le i \le n-2$ ne postoje istoveremno grane (v, x_i) , (u, x_{i+1}). Neka je čvor v povezan sa čvorom x_{n+1} i k čvorova $x_{i1}, x_{i2}, \ldots, x_{ik}$, pri čemu je $2 \le i1 < i2 < \ldots < ik \le n-2$. Tada čvor v ne sme biti vezan ni sa jednim od čvorova $x_{i1+1}, x_{i2+1}, \ldots, x_{ik+1}$, te je $d(v) \le n-2-k$. Zbog toga je je $d(u) + d(v) \le (k+1) + (n-2-k) = n-1$, a ovo je u kontradikciji sa uslovom $d(v) + d(w) \ge n$. Dakle, postoje grane (v, x_i) и (u, x_{i+1}) i od ovih grana se može formirati novi Hamiltonov ciklus koji ne sadrži granu (u, v).
Realizacija: U datom grafu G pronalazimo dugačak put (npr. pomoću DFSa), a onda dodajemo nove grane tako da put produžimo do Hamiltonovog ciklusa. Tako je dobijen veći graf G'. Obično je dovoljno dodati samo nekoliko grana. Međutim, u najgorem slučaju može biti dodata najviše n-1 grana. Dokaz teoreme se primenjuje iterativno polazeći od G', sve dok se ne pronađe Hamiltonov ciklus u G. Ukupan broja koraka za zamenu jedne grane je O(n). A potrebno je zameniti najviše n, pa je složenost algoritma $O(n^2)$.

15. Graf je zadat kao pravougaona mreža dimenzije m × n. Za koje m i n u ovom grafu postoji Hamiltonov ciklus?

1. Ako je barem jedno m, n parno onda





2. Ako su m, n neparni, onda u tom grafu svi ciklusi su parne dužine, a broj čvorova je neparan.

16. Dokazati da ako je G neusmereni bipartitni graf sa neparnim brojem čvorova, tada G ne sadrži Hamiltonov ciklus.
Rešenje:
Neka je G=(V1,V2,E)
Svaka grana e iz E povezuje jedan čvor iz V1 sa čvorom iz V2. Da bi dobijeni put bio ciklus, ako krenemo iz čvora v∈ V1, mi se moramo vratiti u čvor v∈ V1, tj. broj čvorova u Hamiltonovom ciklusu mora biti paran. (spajamo po dva čvora i vraćamo se u čvor polaznik).

17. Graf je bipartitan akko nema ciklus neparne dužine.

18. Dat je graf G = (V,E). Neka su M1 i M2 dva uparivanja u grafu G. Posmatramo novi graf $G_0 = (V, M1 \cup M2)$, tj. graf sa istim skupom čvorova, a skup grana se sastoji od svih grana koje se javljaju u uparivanju M1 ili u uparivanju M2. Dokazati da je graf G_0 bipartitan.

Rešenje:
Neka je $(e_1,e_2,..,e_k)$ niz grana koje formiraju ciklus u G_0. Pokazimo da k mora biti parno.

Vazi da $e_1 \in M1 \cup M2$. Bez smanjenja opstosti, pretpostavimo da $e_1 \in M1$.
Obzirom da e_1, e_2 imaju zajednicki cvor i s obzirom da M1 je uparivanje, onda ne moze da $e_2 \in M1$, te vazi da $e_2 \in M2$.
Slicno $e_3 \in M1$, $e_4 \in M2$...
Indukcijom se pokazuje da vazi $\forall i \leq k, e_i \in M1$ AKKO je ***i*** neparno

Ako je k neparno, onda vazi da $e_k \in M1$
Ali, grane e_k, e_1 imaju zajednicki cvor (jer zatvaraju ciklus), te dobijamo kontradikciju.

Odatle sledi k je parno, tj**. svaki ciklus je parne duzine**, pa je po teoremi iz zadataka 14 graf bipartitan.

19. Neka je G = (V,E) povezan graf sa n >=3 cvorova. Ako za svaka dva nesusedna cvora u, v iz V vazi: d(u)+d(v) >=n onda graf G ima Hamiltonov ciklus (Ore-ova teorema).

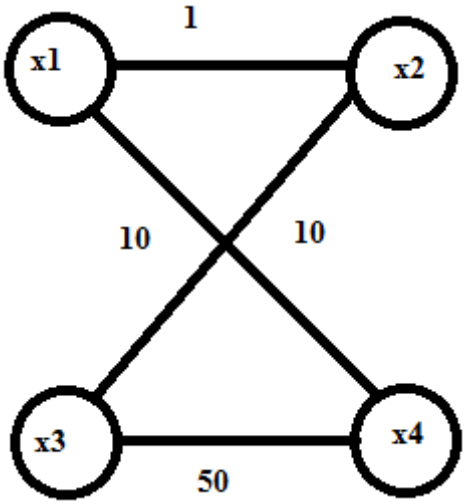
20. Neka je G = (V,E) povezan graf sa n >=3 cvorova u kome je stepen svakog cvora barem n/2. Tada graf sadrzi Hamiltonov ciklus (Dirac-ova teorema).

21. Na konferenciji se pojavilo 111 učesnika. Jedna studija je utvrdila da se svaki student rukovao sa tačno 17 drugih studenata. Da li je ovo moguće?
Rešenje:
Pretpostavimo da je to moguće i da je studija tačna.
Definišemo graf G = (V,E) čiji čvorovi odgovaraju studentima, a grana između dva čvora postoji ako su se oni rukovali.
Znamo da u grafu važi $\sum_{v \in V} d(v) = 2 \mid E \mid$ (suma svih stepena čvorova je jednaka dvostrukom broju grana, jer svaku granu u sumi stepena brojimo po 2 puta).

Stoga bi u grafu iz ovog zadatka trebalo da važi 111*17=2|E| => kontradikcija.

22. Da li je korektan sledeci pohlepni algoritam za nalazenje savrsenog uparivanja minimalne tezine u neusmerenom grafu, za koji znamo da postoji savrseno uparivanje?
(a) Uparivanju dodajemo granu minimalne tezine koja nije incidentna sa nekim od cvorova grana koje se vec nalaze u uparivanju.
(b) Nastavljamo isti postupak sve dok ne dodjemo do |V |/2 grana, gde je |V | broj cvorova u grafu.

Resenje:
Kontraprimer



U ovom grafu, predlozeni algoritam najpre bira granu (x1,x2) tezine 1.
Sledeci iznudjen izbor mora biti neincidentna grana (x3,x4). To svakako nije savrseno uparivanje minimalne tezine.
Savrseno uparivanje minimalne tezine bilo bi M={(x1,x4), (x2,x3)}