Grafovi (flood fill algoritam, artikulacijske tačke, optimalan tok i bipartitno uparivanje)

Podsećanje

Simulacije, BFS, DFS, IDDFS

http://poincare.matf.bg.ac.rs/~jelenagr/2d/Simulacije.pdf

http://poincare.matf.bg.ac.rs/~jelenagr/2d/Grafovi000.pdf

Pohlepni algoritmi za rad sa grafovima

http://poincare.matf.bg.ac.rs/~jelenagr/2d/PohlepniGrafovi.pdf

Binarna i logaritamska stabla

http://poincare.matf.bg.ac.rs/~jelenagr/2d/Stabla.pdf

Algoritam flood fill (poplava)

Flood fill je algoritam koji utvrđuje koliko čvorova je povezano s nekim zadatim čvorom. Takođe se može koristiti i za prebrojavanje povezanih komponenti grafa.

Flood fill algoritam preplavljuje susede datog čvora kako bi pronašao koliko veliki deo grafa može poplaviti. Najjednostavnije se implementira običnim DFS-om ili BFS-om koji se što više šire po neposećenim čvorovima. Zato je složenost ovakve implementacije O(V+E), jer obilazi sve čvorove i sve grane. Ako graf predstavimo matricom susedstva, a ne listom povezanosti, onda složenost je $O(V^2)$. Ako je graf potpun, odnosno $E \sim O(V^2)$, onda su složenosti iste bez obzira da li koristimo matricu susedstva ili listu povezanosti.

Zadatak 01

Đaci IIa igraju se dodavanja lopte. Svaki đak nasumično odabere nekog svog drugara i doda mu loptu, ali nisu svi đaci odeljenja međusbno drugari. Koliko najmanje lopti mora profesor fizičkog dodeliti đacima tako da svako može u igri da dobije loptu?

ULAZ

U prvoj liniji standardnog ulaza zadaje se broj učenika ***n*** (2<=n<=10000) i broj drugarskih parova ***m*** ((2<=n<=100000). Đaci su obelezeni brojevima od 1 do n. Potom se u 2. liniji standardnog ulaza zadaje m parova (ucenici jednog para razdvojeni 1 blanko karakterom, parovi su medjusobno razdvojeni sa po 2 blanko karaktera)

IZLAZ

Broj lopti (ceo broj iz intervala od 1 do n)

TEST PRIMER

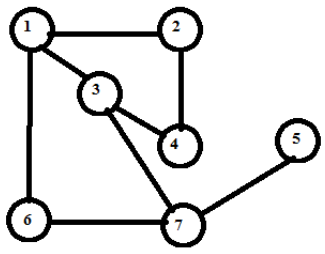
ULAZ

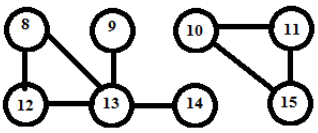
15 17

1 2 1 3 1 6 2 4 2 5 3 4 3 7 5 7 6 7 8 12 8 13 9 13 10 11 10 15 11 15 12 13 13 14

IZLAZ

3





Rešenje:

Predstavicemo graf listom povezanosti vector<vector<int> > graf;

Koristicemo vector<int> posecen; da oznacimo koji smo cvor posetili kako se ne bismo vise puta granali iz istog cvora.

```
#include <iostream>
#include <vector>
using namespace std;

vector<vector<int> > graf;
vector<int> posecen;

void floodFill(int x)
{  posecen[x]=1;
   for(int i=0; i<graf[x].size(); i++)
    if(!posecen[graf[x][i]])
      floodFill(graf[x][i]);
}

int main()
{ int n,m;
 cin >> n >> m;
 vector<int> vi;
 graf.insert(graf.begin(),n,vi);
 posecen.insert(posecen.begin(),n,0);
 for(int i=0;i<m;i++)
 {  int a,b;
    cin >> a >> b;
    graf[a-1].push_back(b-1); //koristimo a-1, b-1 kako bi brojevi bili 0-indeksirani
    graf[b-1].push_back(a-1);
 }

 int rez=0;
 for (int i=0;i<n;i++)
   if (!posecen[i])
   {  rez++;
      floodFill(i);
   }
   cout << rez << endl;
    return 0;
}
```

Graf ucitavamo

```
  cin >> n >> m;
 vector<int> vi;
 graf.insert(graf.begin(),n,vi);
```

Dakle, nakon unosa broja cvorova i broja grama grafa, dodamo na pocetak grafa n puta vektor (vector<int> vi) pozivom metoda

```
   graf.insert(graf.begin(),n,vi);
```

Na taj nacin, mozemo koristiti indeksnu notaciju [ ] kad god zelimo pristupiti spisku suseda bilo kog cvora.

Umesto linija
```
 vector<int> vi;
 graf.insert(graf.begin(),n,vi);
```
mogli smo napisati i
```
   graf.resize(n, vector<int>());
```

Potom naredbom
```
 posecen.insert(posecen.begin(),n,0);
```
ubacujemo n puta 0 na pocetak vektora ***posecen,*** jer zelimo da zabelezimo da jos nismo obisli niti jedan cvor grafa.

U fragmentu koda
```
 for (int i=0;i<n;i++)
   if (!posecen[i])
   {  rez++;
      floodFill(i);
   }
```

Prolazimo redom preko svih cvorova i ako nismo bili u nekom cvoru i (if (!posecen[i])), to znaci da smo nasli novu povezanu komponentu grafa koju citavu obidjemo rekurzivnom funkcijom ***floodFill***.

DISKUSIJA
Složenost navedenog rešenja je O(n+m), jer se svaki čvor obiđe samo jednom i preko svake grane se pokuša ići samo jednom.
Da smo koristili matricu susedstva za predstavljanje grafa, onda bi složenost rešenja bila $O(n^2)$, tačnije $O(n+n^2)$, jer bi u rekurzivnoj funkciji floodFill za svaki cvor morali proveriti da li je povezan sa čvorom za koji je rekurzija pozvana. Tada nam uopšte nije važno koliko je veliko ili malo m (broj grana), jer u matrici susedstva potencijalnih grana ima n^2.

Zadatak 02 (sa hrvatskog takmicenja)

Otkako je stiglo proljeće, sunce je počelo jače sjati pa su ljudi počeli spuštati rolete. Štefica je dežurna baba tračara u svome kvartu i svakodnevno promatra prozore na susjednoj zgradi: zanima je tko spušta rolete i do koje visine ih spušta. Prozor ćemo prikazati kao polje dimenzija 4 x 4, pri čemu su rolete označene zvjezdicama. Štefica će vidjeti svaki prozor u jednoj od sljedećih pet varijanti:

Te varijante redom označuju prozore na kojima: 1) rolete nisu spuštene, 2) rolete su neznatno spuštene, 3) rolete su napola spuštene, 4) rolete su većim dijelom spuštene, 5) rolete su posve spuštene.
Štefica promatra zgradu od **M** katova, a na svakom katu vidi **N** prozora. Prozore stoga možemo smjestiti u tablicu od **M** x **N** polja. Prozori su obrubljeni znakovima #. Napišite program koji pomaže Štefici utvrditi koliko ljudi je spustilo rolete na koju visinu.
ULAZNI PODACI
U prvom retku ulaza nalaze se prirodni brojevi **M** i **N** ($1 \leq \mathbf{M}, \mathbf{N} \leq 100$) iz teksta zadatka.
Sljedeći redci prikazuju Štefičin pogled na zgradu: to je tablica od **M** x **N** prozora od kojih je svaki prikazan kao polje od 4 x 4 znaka u jednoj od pet varijanti opisanih u tekstu zadatka. Između prozora nalaze se rubovi označeni znakovima #. Pogledajte donje primjere za bolje razumijevanje. Može se izračunati da ukupno treba unijeti 5**M** + 1 redaka od po 5**N** + 1 znakova.
IZLAZNI PODACI
Ispišite pet prirodnih brojeva (odvojenih po jednim razmakom): broj prozora od svake varijante (redom kojim su opisane u tekstu zadatka). Zbroj ovih brojeva, naravno, iznosi **M*N**.
PRIMERI TEST PODATAKA

ulaz

```
1 2
###########
#....#****#
#....#****#
#....#....#
#....#....#
###########
```

izlaz

```
1 0 1 0 0
```

ulaz

```
2 3
################
#****#****#****#
#****#****#****#
#****#....#****#
#....#....#****#
################
#....#****#****#
#....#****#....#
#....#....#....#
#....#....#....#
################
```

izlaz

```
1 1 2 1 1
```

Komisijsko resenje (bez flood fill): obilazak matrice uz upotrebu niza od 5 brojaca

```
#include<cstdio>
using namespace std;

char a[505][505];
int sol[5];

int main()
{
	int m, n; scanf("%d%d", &m, &n);
	for(int i=0; i<5*m+1; i++) scanf("%s", a[i]);
	for(int i=0; i<m; i++)
		for(int j=0; j<n; j++)
		{
			int x = 1 + 5*i;
			int y = 1 + 5*j;
			int k = 0;
			for(int l=0; l<4; l++)
				k += (a[x+l][y] == '*');
			sol[k]++;
		}
	for(int i=0; i<5; i++)
      printf("%d%c", sol[i], i==4?'\n':' ');
}
```

Resenje (flood fill strategija)

```
#include <iostream>
using namespace std;
#include <stdio.h>
#include <string.h>
#define Nmax 105

int n,m,zvezdice, gotovo[Nmax*6][Nmax*6];
char a[Nmax*6][Nmax*6];
int sol[5];
```

```
void floodfill(int x, int y)
{
   if( x < 0 || x >= n || y < 0 || y >= m)
      return;
   if( a[x][y] == '#' || gotovo[x][y] )
      return;
   if(a[x][y] == '*')
      zvezdice++;
   gotovo[x][y] = true;
   floodfill(x+1,y);
   floodfill(x,y+1);
   floodfill(x-1,y);
   floodfill(x,y-1);
}

int main()
{
   scanf("%d %d", &n, &m);
   n = 5*n+1;
   m = 5*m+1;
   for(int i = 0; i < n; i++)
      for(int j = 0; j < m; j++)
         scanf(" %c", &a[i][j]);
   memset(gotovo,0,sizeof(gotovo));
   memset(sol,0,sizeof(sol));
   for(int i = 0; i < n; i++)
   {
      for(int j = 0; j < m; j++)
      {
         if(a[i][j] != '#' && !gotovo[i][j])
         {
            zvezdice = 0;
            floodfill(i,j);
            int brojac = zvezdice;
            sol[brojac/4]++;
            /*
            if(brojac == 0)sol[0]++;
            if(brojac == 4)sol[1]++;
            if(brojac == 8) sol[2]++;
            if(brojac == 12)sol[3]++;
            if(brojac == 16) sol[4]++;
            */
         }
      }
   }
   for(int i = 0; i < 5; i++)
      printf("%d ", sol[i]);
   printf("\n");

   return 0;
}
```

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

3. Fabrika pravougaonog oblika sastoji se od n (1<=n<=1000) redova i m (1<=m<=1000) kolona. Svaki deo fabrike ima zidove s nekih strana. Sa standardnog ulaza ucitava se n i m, a potom se ucitava n redova sa po m brojeva. Svaki broj od 0 do 15 opisuje jedan deo fabrika. Binarni zapis broja opisuje s kojih strana tog dela fabrike se nalaze zidovi. Ako je, na primer, dat broj 10, onda je njegov binarni zapis 1010, sto znaci da ima zid sa severa, nema sa istoka, ima s juga i nema sa zapada. Opis ce biti takav da fabrika ima zidove sa spoljne strane. Napisati C program koji ce za svaku prostoriju ispisati njenu velicinu. Velicine prostorija ispisati od vece ka manjoj.

PRIMER
ULAZ
4 5
9 14 11 12 13
5 15 11 6 7
5 9 14 9 14
3 2 14 3 14

IZLAZ
9 4 4 2 1

Resenje:

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

vector<vector<int> > fabrika;
vector<vector<int> > posecen;
int n,m;
int velicina;
const int smerX[]={-1,1,0,0};
const int smerY[]={0,0,-1,1};
const int BitZid[]={8,2,1,4}; //binarni zapis lokacije niza
//niz BitZid je uskladjen sa nizovima smerX, smerY

void floodFill(int x, int y)
{ posecen[x][y]=1;
  velicina++;
  for(int i=0;i<4;i++)
     if(x+smerX[i] >=0 && x+smerX[i]<n  &&
        y+smerY[i] >=0 && y+smerY[i]<m  &&
        !posecen[x+smerX[i]][y+smerY[i]] &&
        !(fabrika[x][y] & BitZid[i]))
           floodFill(x+smerX[i], y+smerY[i]);
}

int main()
{   cin >> n >> m;
    vector<int> vi;
    fabrika.insert(fabrika.begin(),n,vi);
    posecen.insert(posecen.begin(),n,vi);
    int x;
    for(int i=0;i<n;i++)
     for(int j=0;j<m;j++)
     { cin >> x;
```

```
      fabrika[i].push_back(x);
      posecen[i].push_back(0);
    }

    vector<int> resenje;
    for(int i=0;i<n;i++)
      for(int j=0;j<m;j++)
        if(!posecen[i][j])
        { velicina=0;
          floodFill(i,j);
          resenje.push_back(velicina);
        }
      sort(resenje.begin(),resenje.end());
      reverse(resenje.begin(),resenje.end());
      for(int i=0;i<resenje.size();i++)
        cout << resenje[i] << ' ';
      cout << endl;
      return 0;

}
```

Vremenska slozenost O(n*m)

ARTIKULACIJSKE TACKE

Artikulacijske ili rezne tacke su cvorovi koji povezuju neke povezane komponente grafa i njihovim uklanjanjem iz grafa se graf raspada na vise manjih komponenti povezanosti (povecava se broj povezanih komponenti grafa).

Artikulacijske tacke se pronalaze u slozenosti O(E) pomocu DFS obilaska grafa. Pri rekurzivnom obilasku grafa pamtimo u kojim cvorovima smo bili kako ne bismo dva puta obisli isti cvor. Pri rekurzivnom DFS obilasku stvara se DFS stablo i ono se najcesce crta tako da su cvorovi koji su pre obidjeni crtaju iznad onih na koje se rekurzija dalje grana. Tada su nam vazne dve informacije

1. redni broj cvora koji smo nasli (*redniBr*, tzv. discovery time)
2. koji najmanji redniBr moze dohvatiti neki od cvorova iz podstabla za koje se rekurzija grana (tzv. *minBr*)

Cvorove na koje se rekurzija grana zovemo decom cvora koji ih je pozvao, a cvor koji ih je pozvao je roditelj svoje dece.

Pomocu brojeva *redniBr* i *minBr* moze se ustanoviti koji cvorovi su atrikulacijske tacke. Cvor je artikulacijska tacka ako i samo ako u DFS stablu cvor X ima neko dete Y tako da redniBr(X) <= miniBr(Y)
Dakle, cvor je artikulacijska tacka ako niko od njegove neposredne dece ne moze se spustajuci se direktnim granama u DFS stablu i penjuci se povratnim granama doci iznad njega.

Pri trazenju artikulacijskih tacaka svejedno je iz kog cvora krecemo, jer iako postoji vise DFS stabala za jedan graf, dobijamo iste artikulacijske tacke.

Zadatak 04: Neprijatelj je napao drzavu sa n gradova (1<=n<=100000) i m puteva (1<=m<=1000000) izmedju gradova. Gradovi su medjusobno povezani. Kazemo da grad je vazan ako bi njegovim osvajanjem neprijatelj razdvojio ostale gradove. Napisati C program koji ce ispisati sve vazne gradove poredjane po velicini. Ulazni podaci su n, m, a potom m puta po dva broja a i b (0<=a,b<=n-1) koji predstavljaju dva 0-indeksirana grada povezana putem.
PRIMER
ULAZ
11 13
0 1 0 3 1 2 2 3 2 4
2 5 4 5 5 6 6 6 6 8
8 9 8 10 7 8

IZLAZ
2 5 6 8

Resenje:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<vector<int> > graf; //lista povezanosti, suseda
vector<int> redniBr;
vector<int> artTacke; //RESENJE
int rBr=1;

int DFS(int roditelj, int cvor)
{redniBr[cvor]=rBr;
 int minBr=rBr; //za sada, dostize samo sebe
 int kuda, brojGrananja=0, t, maxt=0;
 for(int i=0;i<graf[cvor].size();i++)
 {  kuda=graf[cvor][i];
    if(kuda==roditelj) continue; //odatle smo dosli
    if(redniBr[kuda])  //ako je vec posecen
      minBr=min(minBr, redniBr[kuda]);
    else {
      rBr++;
      brojGrananja++;
      t=DFS(cvor, kuda);
      maxt=max(maxt,t);
      minBr=min(minBr,t);
    }
 }
  if(redniBr[cvor]==1){ //ako je pocetni
      if(brojGrananja>1) artTacke.push_back(cvor);
  } else if (maxt >= redniBr[cvor]) //&& graf[cvor].size()>1
     artTacke.push_back(cvor);
     return minBr; //do toliko najvise se mogu popeti
}

int main()
{  int n,m; cin >>n >>m;
vector<int> vi;
graf.insert(graf.begin(),n,vi);
int a,b;
for(int i=0;i<m;i++) {
 cin >> a >> b;
 if (a==b) continue;  //necemo cikluse
 graf[a].push_back(b);
 graf[b].push_back(a);
}

redniBr.insert(redniBr.begin(),n,0);  //0 znaci neposecen
DFS(-1,0);   //roditelj, trenutni cvor
sort(artTacke.begin(), artTacke.end());
for(int i=0;i<artTacke.size();i++)
 cout << artTacke[i] << ' ';
    return 0;
```

}

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.

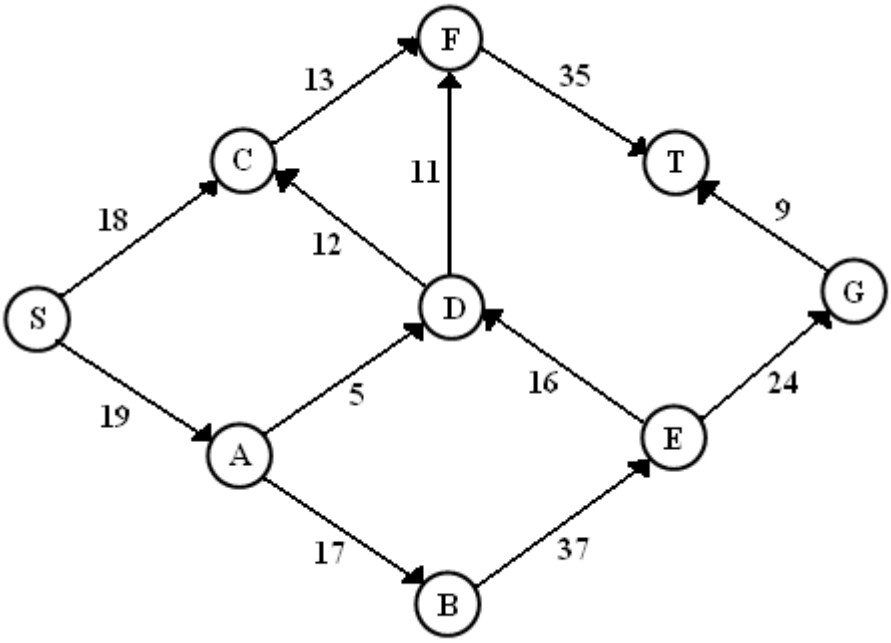
Povećavajući put je uopštenje alternirajućeg puta i ima isti smisao za transportne mreže kao alternirajući put za bipartitivno uparivanje.
Teorema: Tok kroz mrežu je optimalan akko u odnosu na njega ne postoji povećavajući put.
Teorema o povećavajućem putu neposredno se transformiše u algoritam. Polazi 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.

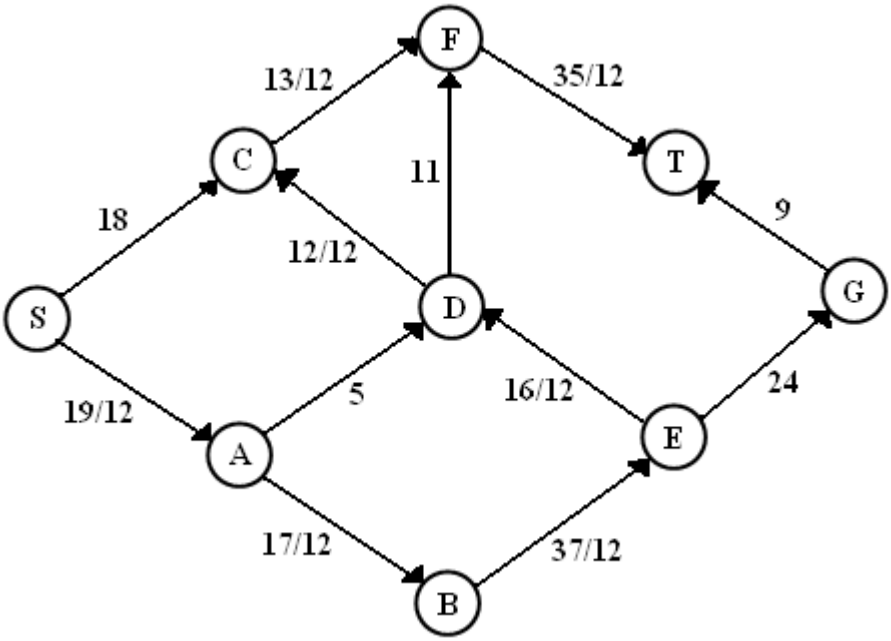
Zbog rešavanja zadatka uz pomoć rezidualnog grafa, za svaku cev postavlja se imaginarna u kojoj voda prolazi u suprotnom smeru i količina vode koja prolazi kroz imaginarnu cev je samo suprotna po znaku.

1. Pronađite maksimalni protok datog grafa.

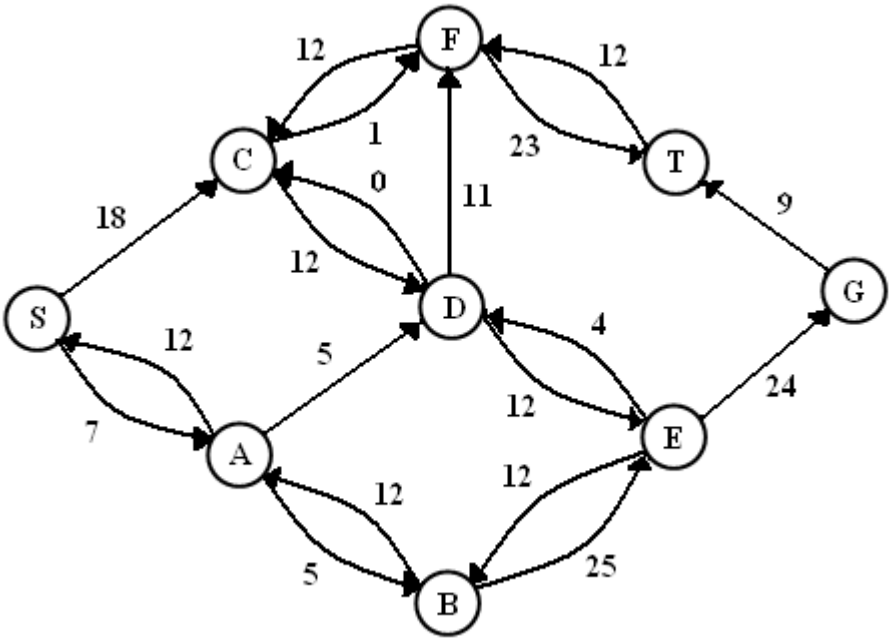


Prvi pronađeni put je SABEDCFT(pretraga po dubini)
Nosivosti grana tog puta su: 19,17,37,16,12,13,35

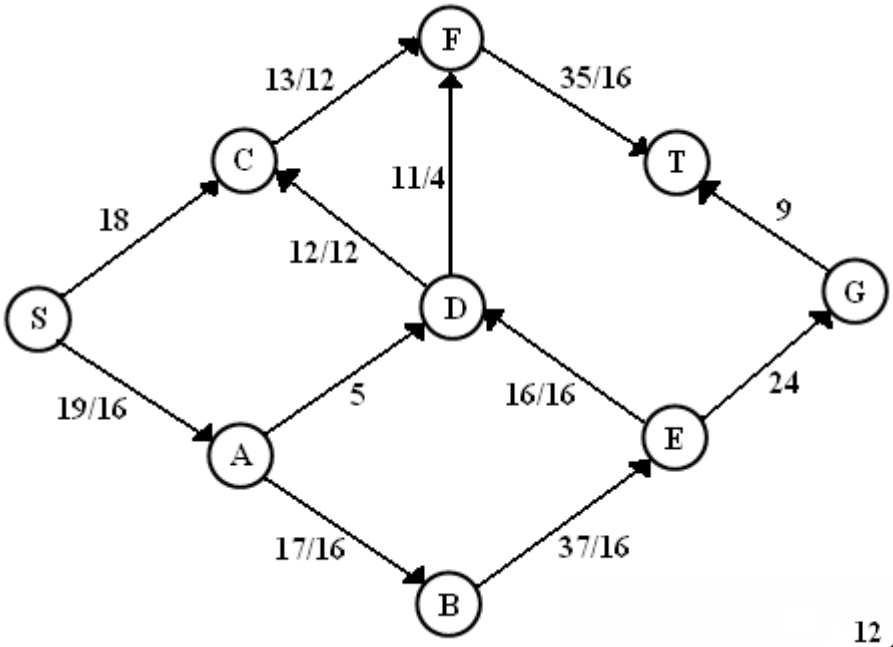
Minimum je 12 i ta se količina prva propušta kroz grane tog puta:



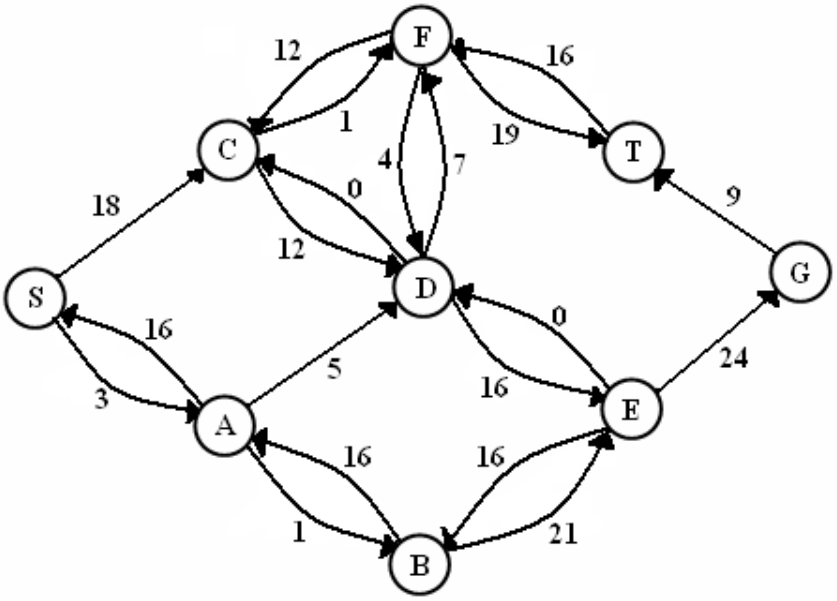
Sa imaginarnim cevima:



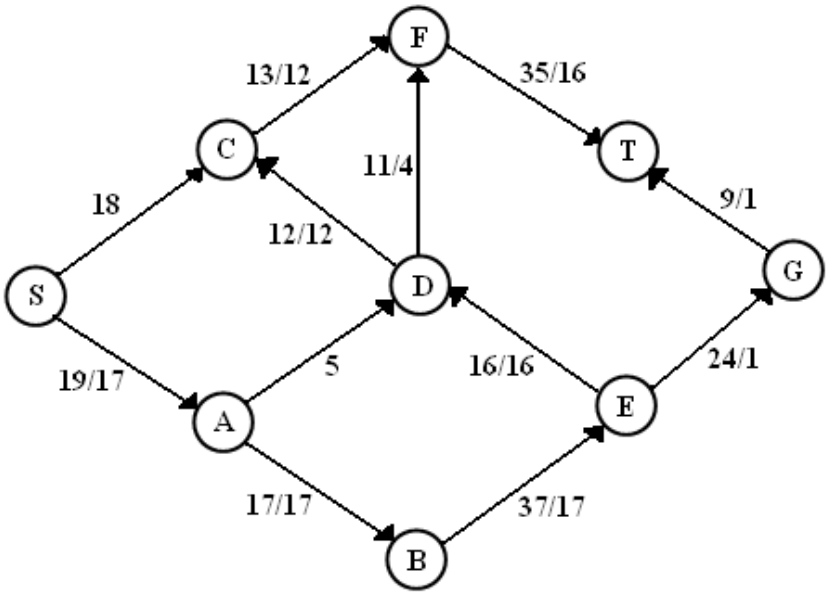
Drugi pronađeni put je SABEDFT
Nosivosti grana tog puta su: 7,5,25,4,11,23
Minimum je 4 i ta kolicina se sledeca propusta:

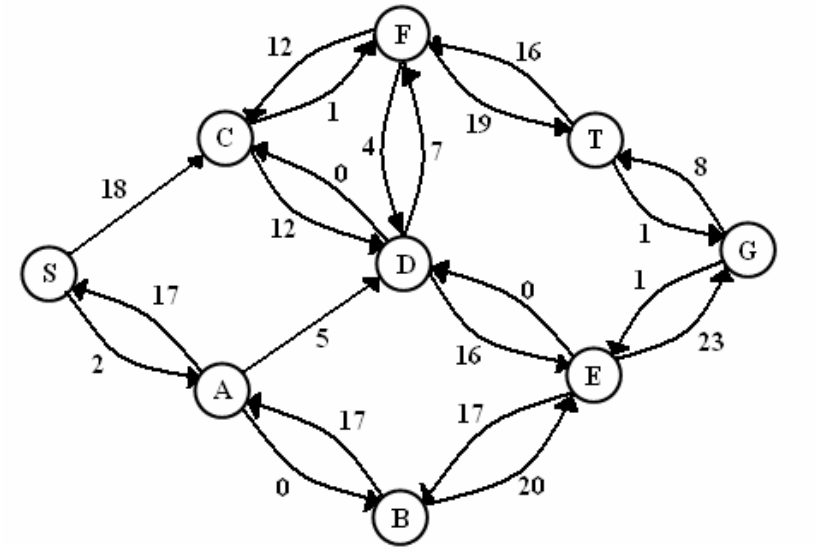


Sa imaginarnim cevima:



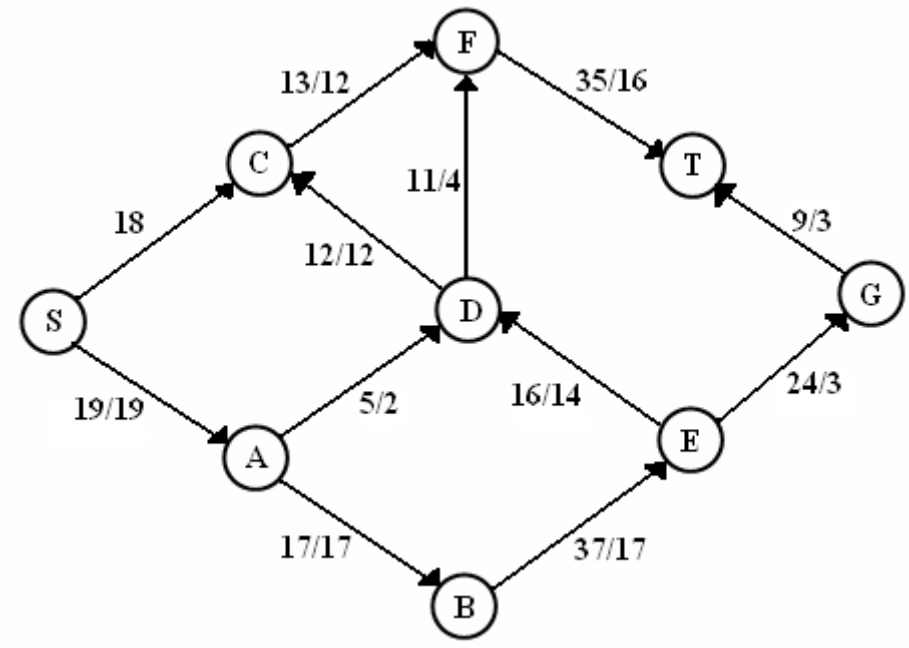
Treci pronadjeni put je SABEGT
Nosivosti grana tog puta su: 3,1,21,24,9
Minimum je 1 i ta se kolicina sledeca propusta:



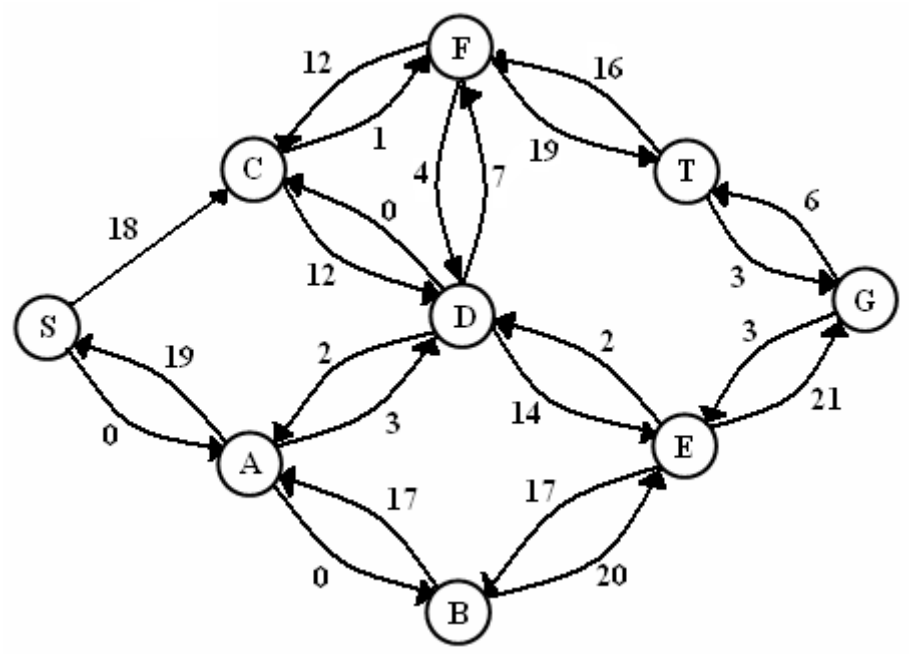


Sa imaginarnim cevima:

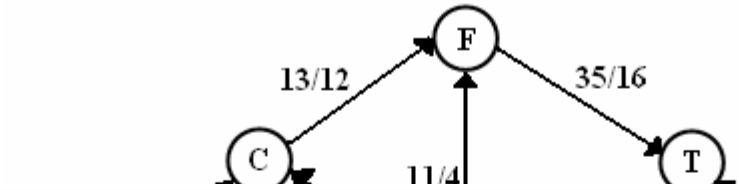
Četvrti pronađeni put je SADEGT
Nosivosti grana tog puta su: 2,5,16,23,8
Minimum je 2 i ta se kolicina sledeca propusta:



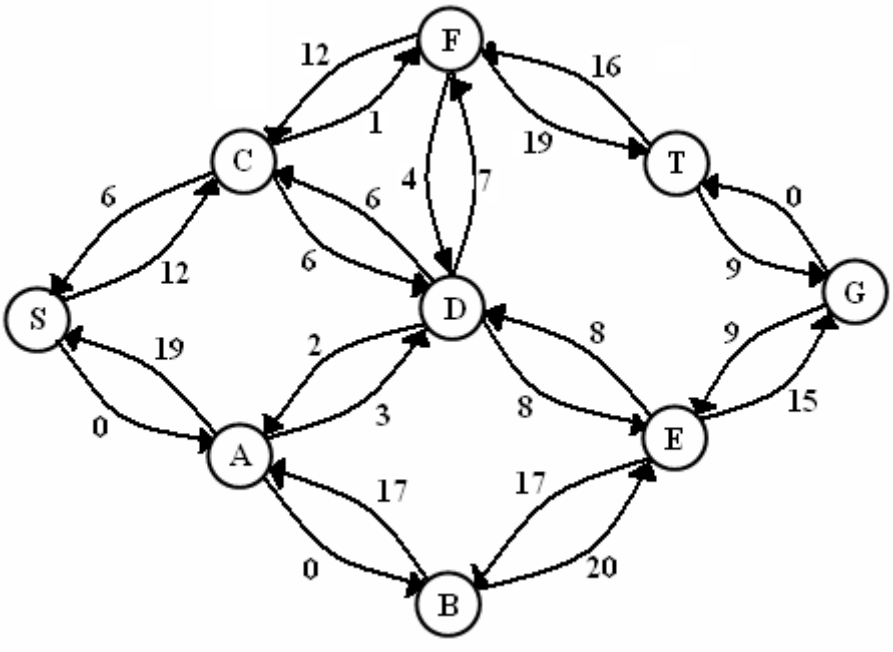
Sa imaginarnim cevima:



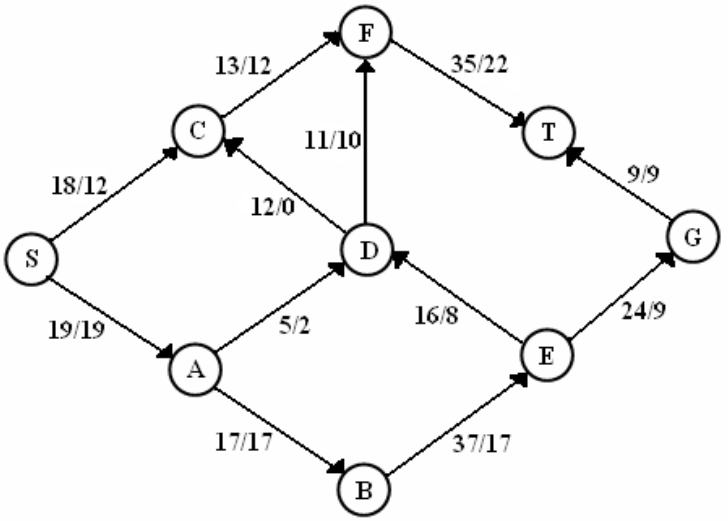
Peti pronadjeni put je SCDEGT
Nosivosti grana tog puta su: 18,12,14,21,6
Minimum je 6 i ta se kolicina sledeca propusta:

Sa 

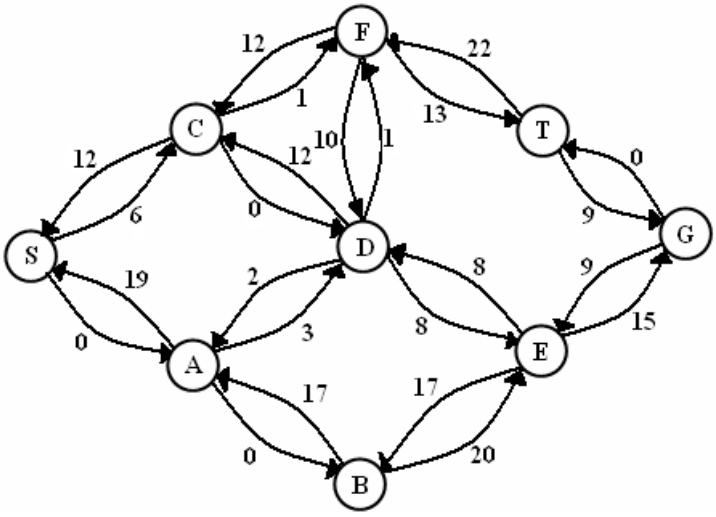
imaginarnim cevima:



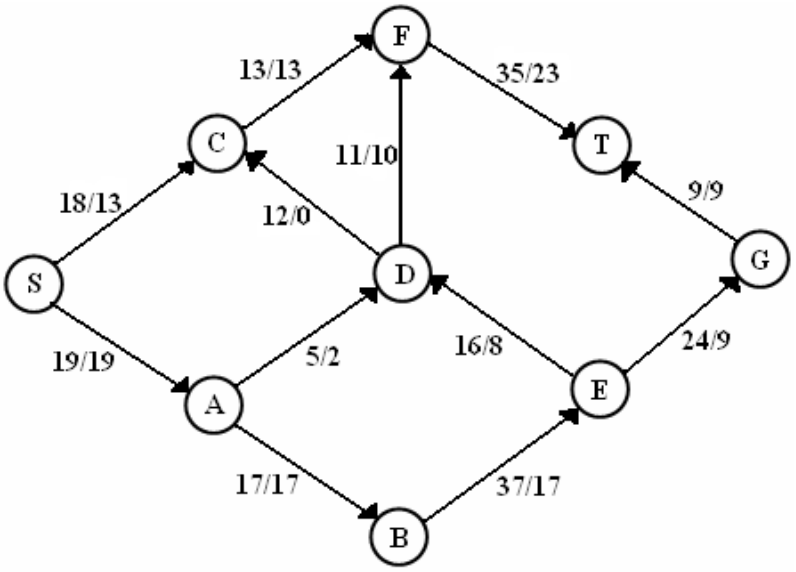
Šesti pronadjeni put je SCDFT
Nosivosti grana tog puta su: 12,6,7,16
Minimum je 6 i ta se kolicina sledeca propusta:



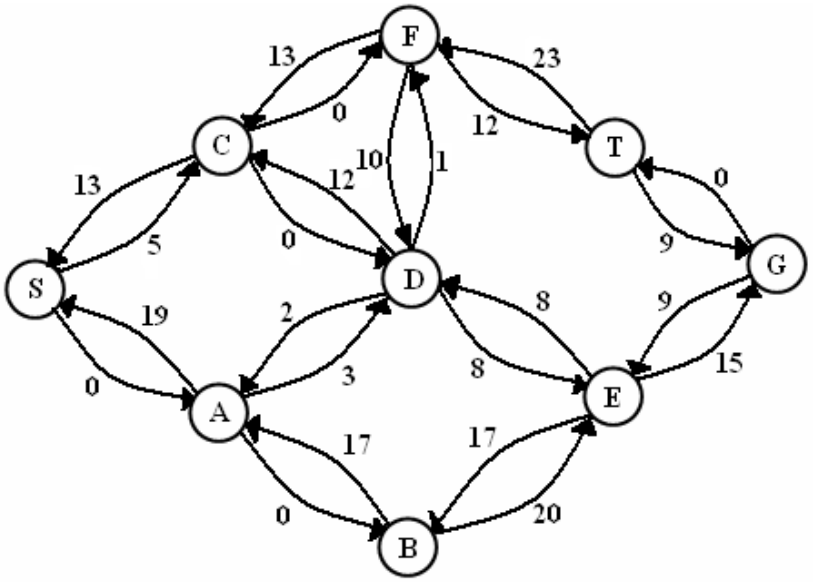
Sa imaginarnim cevima:



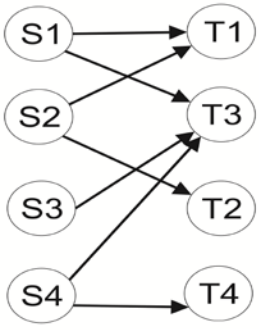
Sedmi pronadjeni put je SCFT
Nosivosti grana tog puta su: 6,1,13. Minimum je 1 i ta se kolicina sledeca propusta:

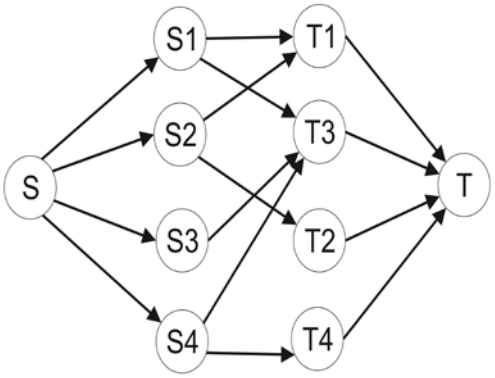


Sa imaginarnim cevima:

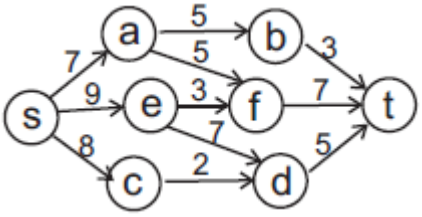


Maksimalni protok u ovom primeru je 32 i predstavlja zbir minimuma pri svakom novom određivanju mogućeg puta.
12+4+1+2+6+6+1=32
Kada graf ima vise izvora i(ili) vise prijemnika zadatak se resava tako sto se postavi novi izvor iz kog se dolazi direktno u sve izvore i(ili) novi prijemnik u koji se dolazi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

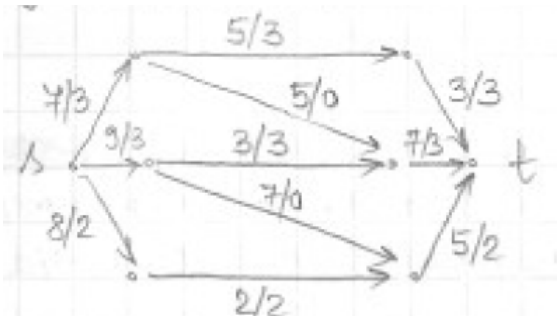




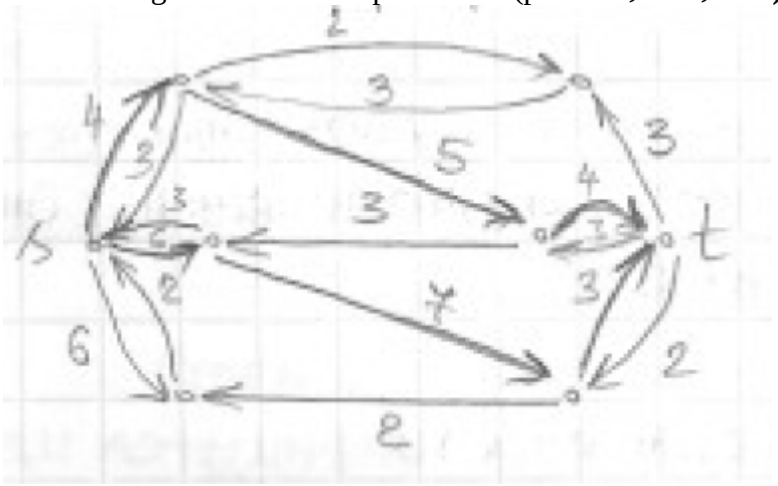
2. (primer iz CLRS knjige) Dat je usmeren graf G = (V,E) sa dva izdvojena čvora *s* i *t*. Granama grafa pridruženi su brojevi – njihovi kapaciteti. Odrediti optimalni tok kroz mrežu.



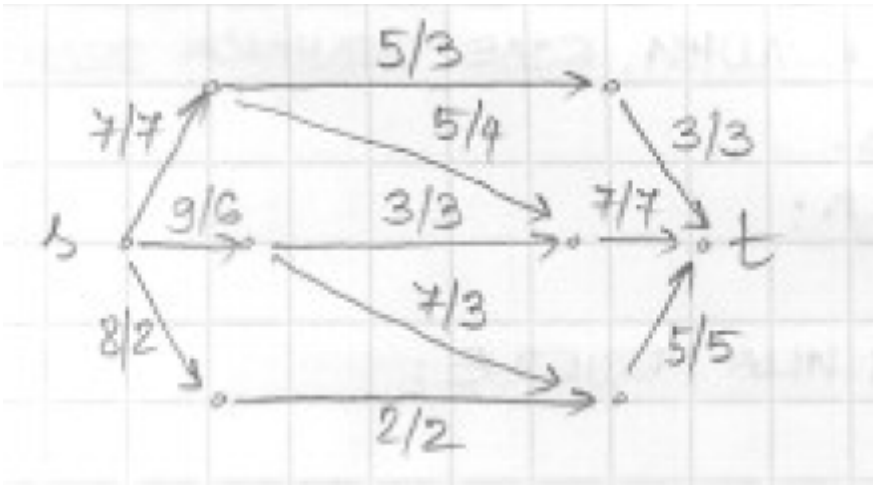
Resenje:
Nađemo neki tok. Svaka grana je oznacena sa a/b gde a =kapacitet, b=trenutni tok.
Možemo da krenemo puštanjem protoka 3 kroz grane kapaciteta 7,5,3 (put **sabt)**
Možemo da krenemo puštanjem protoka 3 kroz grane kapaciteta 9,3,7 (put **seft)**
Možemo da krenemo puštanjem protoka 2 kroz grane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

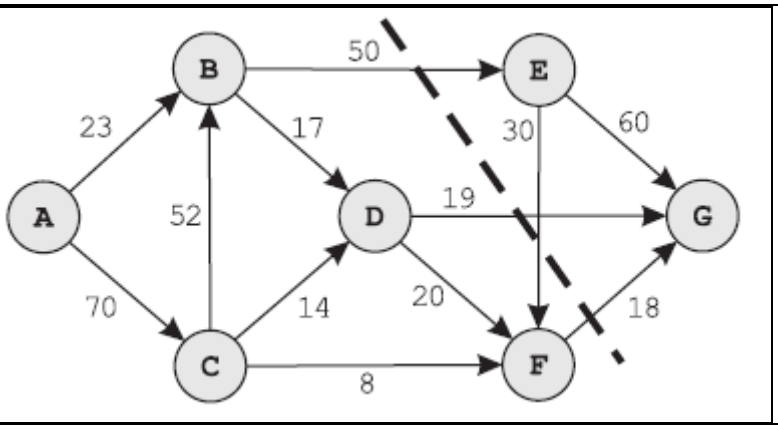
Rešenje:
Formulacija:
Čvor – ugao
Grane – trotoari i pešački prelazi koji povezuju uglove.
Polazni čvor s – kuća profesora Krstića
Izlazni čvor t – škola
Sve grane imaju težninu 1, jer najviše jedno dete može hodati duž grana grafa (trotoar, pešački prelaz)

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.

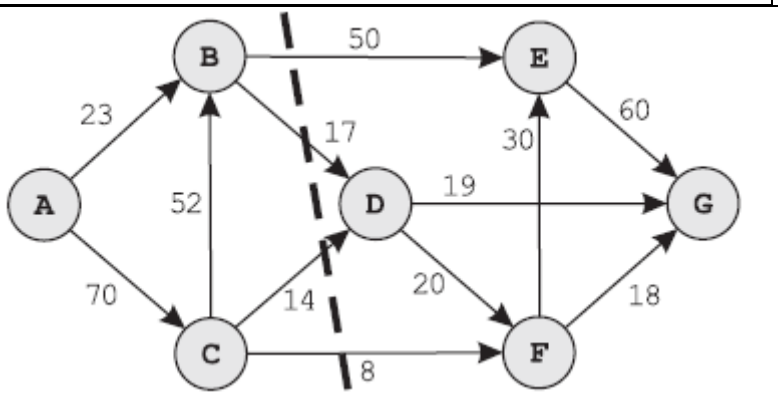
5.
Maksimalan tok kroz transportnu mrežu == minimalna suma težine grana čijim uklanjenjem se ne može doći od izvor s do ponora t

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.

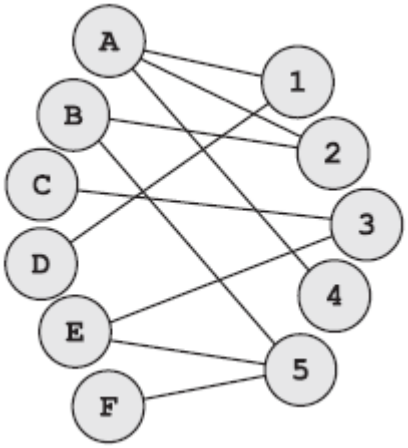
Drugi način za utvrđivanja koje grane čine *Minimum cut* jest pokušati izbaciti svaku granu (od lakših prema težim) i ako se *Network flow* smanji za težinu izbačene grane, znamo da je ta grana deo *Minimum cut-a* i valja nastaviti bez nje.

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.

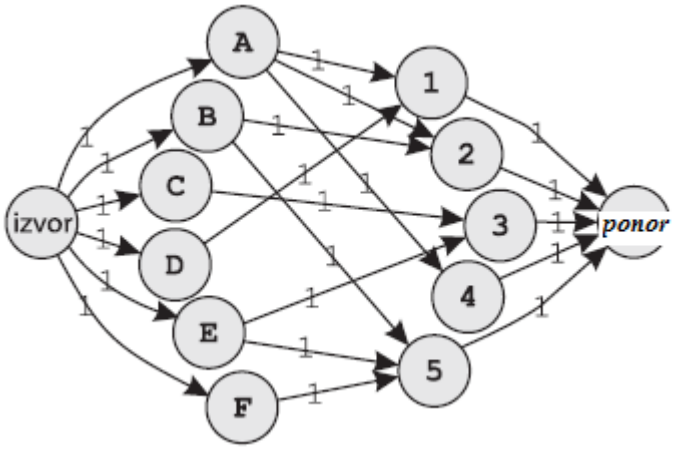
Računari Y1, Y2, ..., Yn su izvori, a X je ponor. Budući da ima više izvora, moramo dodati virtuelan izvor s granama beskonačne težine do izvora Y1, Y2, ..., Yn.
Kako je polazni graf neusmeren, onda svaka grana od čvora A do čvora B težine C se razmatra u dva pojavna oblika, tj. dve usmerene grane, od 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.

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 (Ford-Fulkerson) algorutam na ovoj mrezi odredjuje resenje problema sa matricama, gde je tok kroz (xij ,vi) = (xij, uj) =m_{ij}