Svi najkraći putevi

Razmotrimo problem izračunavanja najkraćih puteva između svaka dva čvora u težinskom grafu. Težine grana mogu biti negativne, ali u grafu ne sme postojati ciklus negativne težine.

Problem: Dat je težinski graf $G = (V, E)$ (usmereni ili neusmereni). Pronaći puteve minimalne dužine između svaka dva čvora u grafu.

Ponovo, pošto govorimo o najkraćim putevima, težine grana zvaćemo dužinama. Ovaj problem nazivamo *problem nalaženja svih najkraćih puteva* (eng. all pairs shortest path). Za početak ćemo se zadovoljiti nalaženjem dužina svih najkraćih puteva, umesto samih najkraćih puteva. Pretpostavimo da je graf usmeren; sve što će biti rečeno važi i za neusmerene grafove.

Kao i obično, pokušajmo sa direktnim induktivno-rekurzivnim pristupom. Može se koristiti indukcija po broju grana ili broju čvorova u grafu. Označimo sa $d(u, v)$ dužinu grane (u, v), a sa $r(u, v)$ dužinu trenutno poznatog najkraćeg puta od čvora u do čvora v

Razmotrimo najpre indukciju po broju grana. Pretpostavimo da smo iz grafa G uklonili granu (u, w) i da smo rešili problem na preostalom grafu G'. Kako se menjaju najkraći putevi u grafu posle dodavanja nove grane (u, w) u graf? Nova grana može pre svega da predstavlja kraći put od do sada pronađenog najkraćeg puta između čvorova u i w. Dakle, potrebno je proveriti da li važi $d(u, w) < r(u, w)$ i ako važi postaviti vrednost $r(u, w)$ na $d(u, w)$. Pored toga, može se promeniti i najkraći put između proizvoljna druga dva čvora v_1 i v_2. Da bi se ustanovilo ima li promene, treba sa prethodno poznatom najmanjom dužinom puta od v_1 do v_2 uporediti zbir dužina najkraćeg puta od v_1 do u, dužine grane (u, w) i dužine najkraćeg puta od w do v_2. Dakle, ako je $r(v_1, u) + d(u, w) + r(w, v_2) < r(v_1, v_2)$ potrebno je novu vrednost $r(v_1, v_2)$ postaviti na $r(v_1, u) + d(u, w) + r(w, v_2)$. Ukupno, za svaku novu granu potrebno je izvršiti $O(|V|^2)$ provera, pa je složenost ovakvog algoritma u najgorem slučaju $O(|E| \cdot |V|^2)$. Pošto je broj grana najviše $O(|V|^2)$, složenost ovog algoritma iznosi $O(|V|^4)$.

Razmotrimo sada indukciju po broju čvorova. Pretpostavimo da smo iz grafa G uklonili čvor u i da smo između svaka druga dva čvora u grafu pronašli najkraći put. Kako se menjaju najkraći putevi u grafu ako se u graf doda novi čvor u? Potrebno je najpre pronaći najkraće puteve od čvora u do svih ostalih čvorova, i najkraće puteve od svih ostalih čvorova do čvora u. Pošto su dužine najkraćih puteva koji ne sadrže u već poznate, najkraći put od u do w možemo da pronađemo na sledeći način. Potrebno je da odredimo samo prvu granu na tom putu. Ako je to grana (u, v), onda je dužina najkraćeg puta od u do w jednaka zbiru dužine grane (u, v) i dužine najkraćeg puta od v do w, koji je već poznat. Potrebno je dakle da uporedimo ove zbirove za sve grane koje polaze iz čvora u, i da među njima izaberemo najmanji:

$$r(u, w) = \min_{(u,v)\in E}\{d(u, v) + r(v, w)\}.$$

Najkraći put od proizvoljnog čvora w do čvora u može se pronaći na sličan način. Ove provere su u najgorem slučaju (kada iz čvora u polazi/u njega dolazi $\Theta(|V|)$ grana) jednak $O(|V|^2)$. Ali to sve nije dovoljno. Ponovo je potrebno da za svaki par čvorova proverimo da li između njih postoji novi kraći put kroz novi čvor u. Za svaka dva čvora v_1 i v_2, da bi se ustanovilo ima li promene, treba sa prethodno poznatom najmanjom dužinom puta od v_1 do v_2 uporediti zbir dužine najkraćeg puta od v_1 do u i dužine najkraćeg puta od u do v_2. Ako važi $r(v_1, u) + r(u, v_2) < r(v_1, v_2)$ vršimo ažuriranje vrednosti $r(v_1, v_2)$. To je ukupno $O(|V|^2)$ provera i sabiranja posle dodavanja svakog novog čvora, pa je složenost ovakvog algoritma u najgorem slučaju $O(|V| \cdot |V|^2) = O(|V|^3)$. Ukupan broj koraka za određivanje dužina najkraćih puteva od i do novododatog čvora je takođe $O(|V|^3)$. Ispostavlja se da je indukcija po broju čvorova efikasnija nego indukcija po broju grana.

Razmotrimo implementaciju algoritma za računanje svih najkraćih puteva u grafu zasnovanog na indukciji po čvorovima, u slučaju kada je graf zadat matricom povezanosti.

```
const int INF = numeric_limits<int>::max();

// odredjujemo sve najkrace puteve indukcijom po broju cvorova
vector<vector<int>> sviNajkraciPutevi(
          const vector<vector<int>>& matricaPovezanosti) {
  int brojCvorova = matricaPovezanosti.size();
  vector<vector<int>> najkraciPut(brojCvorova);
  for (int i = 0; i < brojCvorova; i++)
    najkraciPut[i].resize(brojCvorova, INF);

  // dodajemo jedan po jedan cvor
  for (int i = 0; i < brojCvorova; i++) {
    najkraciPut[i][i] = 0;

    // odredjujemo najkrace puteve od cvora i do svih prethodnih
    // cvorova j
    for (int j = 0; j < i; j++) {
      // pretpostavljamo da je direktno rastojanje najkrace
      najkraciPut[i][j] = matricaPovezanosti[i][j];
      // proveravamo da li je mozda bolji put od i do j koji vodi preko
      // nekog prethodnog cvora k
      for (int k = 0; k < i; k++)
        // ako postoji grana od i do k i put od k do j
        if (matricaPovezanosti[i][k] != INF && najkraciPut[k][j] != INF &&
```

```
            matricaPovezanosti[i][k] + najkraciPut[k][j] < najkraciPut[i][j])
          najkraciPut[i][j] = matricaPovezanosti[i][k] + najkraciPut[k][j];
    }

    // odredjujemo najkrace puteve do cvora i od svih prethodnih
    // cvorova j
    for (int j = 0; j < i; j++) {
      // pretpostavljamo da je direktno rastojanje najkrace
      najkraciPut[j][i] = matricaPovezanosti[j][i];
      // proveravamo da li je mozda bolji put od cvora j do i koji
      // vodi preko nekog prethodnog cvora k
      for (int k = 0; k < i; k++)
        // ako postoji grana od i do k i put od k do j
        if (najkraciPut[j][k] != INF && matricaPovezanosti[k][i] != INF &&
            najkraciPut[j][k] + matricaPovezanosti[k][i] < najkraciPut[j][i])
          najkraciPut[j][i] = najkraciPut[j][k] + matricaPovezanosti[k][i];
    }

    // popravljamo rastojanja od prethodnih cvorova j do prethodnih
    // cvorova k, analizirajuci puteve koji vode preko cvora i
    for (int j = 0; j < i; j++)
      // ako postoji put od j do i i ako postoji put od i do k
      for (int k = 0; k < i; k++)
        if (najkraciPut[j][i] != INF && najkraciPut[i][k] != INF &&
            najkraciPut[j][i] + najkraciPut[i][k] < najkraciPut[j][k])
          najkraciPut[j][k] = najkraciPut[j][i] + najkraciPut[i][k];
  }

  return najkraciPut;
}

int main() {

  // broj cvorova u grafu
  int n;
  cin >> n;
  // matrica susedstva grafa koji sadrzi n cvorova
  vector<vector<int>> M(n);
  for (int i = 0; i < n; i++) {
    M[i].resize(n);
    for (int j = 0; j < n; j++){
      cin >> M[i][j];
      // ako je uneto -1, tezinu grane postavljamo na +beskonacno
      // pretpostavljamo da u grafu ne postoji grana negativne tezine
      if (M[i][j] == -1)
        M[i][j] = INF;
```

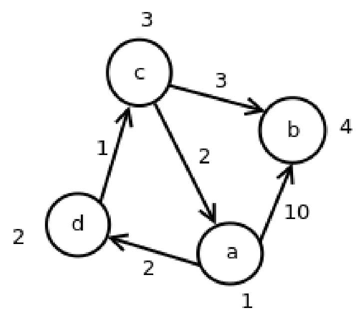
```
    }
  }

  // racunamo najkrace puteve
  auto duzinaPuta = sviNajkraciPutevi(M);
  // stampamo najkrace puteve
  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++)
      if (duzinaPuta[i][j] != INF)
        cout << duzinaPuta[i][j] << "\t";
      else
        cout << "-\t";
    cout << endl;
  }
  cout << endl;
  return 0;
}
```

Međutim, postoji još jednostavnija induktivna konstrukcija za rešavanje ovog problema. Ideja na kojoj se zasniva ovaj algoritam je da se ne menja broj čvorova ili grana u grafu, nego da se uvedu ograničenja na tip dozvoljenih puteva. Indukcija se izvodi po opadajućem broju takvih ograničenja, tako da na kraju dolaze u obzir svi mogući putevi (kada broj ograničenja postane 0). Numerišimo čvorove grafa na proizvoljan način brojevima od 1 do $|V|$. Put od čvora u do čvora w zove se *k-put* ako su redni brojevi svih čvorova na tom putu (izuzev u i w) manji ili jednaki od k. Specijalno, 0-put od čvora u do čvora v se sastoji samo od direktne grane od čvora u do čvora v (pošto se ni jedan drugi čvor ne može pojaviti na tom putu).

Razmotrimo primer grafa sa slike 1. Neka su redni brojevi čvorova a, d, c, b redom 1, 2, 3, 4. Put a, b od čvora a do čvora b dužine 10 je 0-put jer se sastoji iz samo jedne grane (nema usputnih čvorova na putu). Put a, d, c, b dužine 6 je 3-put jer su redni brojevi svih čvorova na tom putu manji ili jednaki 3 (istovremeno je i 4-put), ali nije npr. 2-put. Slično, put c, a, b je 1-put jer su redni brojevi svih čvorova (u ovom slučaju jednog jedinog čvora) na putu manji ili jednaki 1 (ovaj put je, takođe, i 2-put, 3-put i 4-put). Primetimo da je najkraći 0-put (istovremeno i najkraći 1-put i najkraći 2-put) od čvora a do čvora b put a, b dužine 10, dok je najkraći 3-put (i istovremeno najkraći 4-put) jednak a, d, c, b dužine 6.

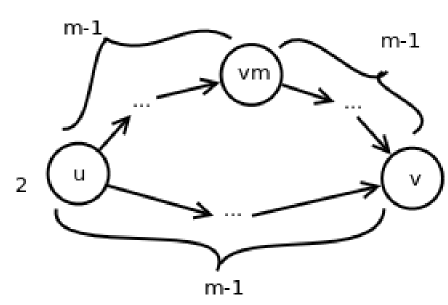
Induktivna hipoteza: Umemo da odredimo dužine najkraćih $k-1$-puteva između svaka dva čvora za $k < m$.

Baza indukcije je slučaj $m = 1$, kad se razmatraju samo direktne grane i rešenje je očigledno: vrednost je jednaka dužini grane ako ona postoji, a ∞ ako grana ne postoji. Pretpostavimo da je induktivna hipoteza tačna i da hoćemo da je proširimo na $k \leq m$. Jedini novi putevi koje treba da razmotrimo su m-putevi. Treba da pronađemo najkraće m-puteve između svaka dva čvora. Neka je v_m



Slika 1: Ilustracija k-puteva u usmerenom težinskom grafu.

čvor sa rednim brojem m. Proizvoljan najkraći m-put sadrži čvor v_m najviše jednom. Naime, pretpostavka je da u grafu ne postoji ciklus negativne dužine, pa najkraći put ne može dva puta da prođe kroz čvor v_m — u protivnom bi se put mogao skratiti izbacivanjem ciklusa od grana između prvog i drugog pojavljivanja čvora v_m. Najkraći m-put između čvora u i čvora v je ili najkraći $(m-1)$ put od čvora u do čvora v ili se sastoji od najkraćeg $(m-1)$-puta od čvora u do čvora v_m, i najkraćeg $(m-1)$-puta od čvora v_m do čvora v jer su čvorovi na m-putu od čvora u do čvora v_m i od čvora v_m do čvora v numerisani brojevima manjim ili jednakim od m (slika 2). Prema induktivnoj hipotezi mi već znamo dužine najkraćih k-puteva za $k < m$. Dakle, da bismo pronašli dužinu najkraćeg m-puta od čvora u do čvora v dovoljno je da saberemo ove dve dužine i zbir uporedimo sa dužinom najkraćeg $(m-1)$-puta od čvora u do čvora v. Do ovog algoritma su nezavisno došli i objavili ga Robert Flojd i Stiven Varšal i poznat je pod nazivom *Flojd-Varšalov algoritam.* Napomenimo da je nekoliko godina pre njih do istog algoritma došao i Bernard Roj, međutim, taj rezultat je prošao nezapaženo. Ipak, negde u literaturi se ovaj algoritam pominje i pod nazivom Roj-Varšalov ili Roj-Flojdov algoritam. Flojd-Varšalov algoritam je za konstantan faktor brži od algoritma zasnovanog na primeni indukcije po broju čvorova i lakše ga je realizovati.



Slika 2: Ilustracija k-puteva u usmerenom težinskom grafu.

Razmotrili smo na koji način možemo izračunati dužine najkraćih puteva između svaka dva čvora u grafu. Same najkraće puteve možemo rekonstruisati na sledeći način: koristimo matricu *put* u kojoj ćemo na poziciji (i, j) pamtiti maksimalnu vrednost k takvu da čvor v_k pripada najkraćem putu od čvora i do čvora j, a ako je najkraći put direktna grana od i do j onda će vrednost $put[i][j]$ biti postavljena na i. Ako se prilikom traženja najkraćeg puta između čvorova i i j nađe kraći put koji vodi preko čvora sa oznakom k, ažuriraćemo vrednost $put[i][j]$ na k. Ispisivanje najkraćeg puta između čvorova i i j svešćemo na ispisivanje najkraćeg puta od čvora i do čvora sa rednim brojem $put[i][j]$ za kojim sledi najkraći put od čvora sa rednim brojem $put[i][j]$ do čvora j.

U kodu koji sledi pretpostavićemo da u grafu ne postoji grana negativne težine. Graf će biti zadat matricom povezanosti jer ona skoro u potpunosti kodira dužine 0-puteva u grafu. Naime jedino je potrebno u poljima matrice koja odgovaraju parovima čvorova između kojih ne postoji grana postaviti umesto vrednosti -1 vrednosti ∞. Kao numeraciju čvorova iskoristićemo njihov indeks.

```
const int INF = numeric_limits<int>::max();

// funkcija koja stampa najkraci put od cvora i do cvora j
// bez ispisivanja cvora j
void odstampajPut(vector<vector<int>> put, int i, int j){

  if (put[i][j] == -1)
    return;
  // put od i do j odgovara direktnoj grani (i,j)
  if (put[i][j] == i)
    cout << i << " - ";
  else{
    // stampamo put od cvora i do cvora k, gde je k maksimalni redni broj cvora
    // koji pripada tom putu, pa zatim put od cvora k do cvora j
    odstampajPut(put, i, put[i][j]);
    odstampajPut(put, put[i][j], j);
  }
}

// funkcija koja stampa najkrace puteve izmedju svaka dva cvora u grafu
void odstampajPuteve(vector<vector<int>> duzinaPuta,
                     vector<vector<int>> put, int brojCvorova){
  cout << "Matrica najkracih rastojanja jednaka je: " << endl;
  for (int i = 0; i < brojCvorova; i++){
    for (int j = 0; j < brojCvorova; j++)
      if (duzinaPuta[i][j] != INF)
        cout << duzinaPuta[i][j] << "\t";
      else
        cout << "-\t";
    cout << endl;
  }
```

```
  cout << endl;
  for (int i = 0; i < brojCvorova; i++)
    for (int j = 0; j < brojCvorova; j++)
      if (i != j && put[i][j] != -1){
        cout << "Najkraci put od cvora " << i
             << " do cvora " << j << " je: ";
        odstampajPut(put,i,j);
        cout << j << endl;
      }
}

// funkcija koja racuna najkrace puteve izmedju svaka dva cvora
void sviNajkraciPutevi(const vector<vector<int>> &matricaPovezanosti) {

  int brojCvorova = matricaPovezanosti.size();
  // inicijalizujemo duzine najkracih puteva na duzine direktnih grana
  // a ako direktna grana ne postoji na vrednost +beskonacno
  vector<vector<int>> najkraciPut = matricaPovezanosti;
  vector<vector<int>> put(brojCvorova);

  // inicijalizujemo najkrace puteve
  for (int i = 0; i < brojCvorova; i++){
    put[i].resize(brojCvorova);
    for (int j = 0; j < brojCvorova; j++)
      // ako postoji direktna grana od cvora i do cvora j
      // pamtimo tu informaciju
      if (matricaPovezanosti[i][j] != INF)
        put[i][j] = i;
      // inace za i<>j postavljamo da je maksimalna oznaka cvora
      // na trenutnom putu -1
      else if (i != j)
        put[i][j] = -1;
      // slucaj kada su razmatramo put od cvora do njega samog
      else
        put[i][j] = 0;
  }

  // proveravamo da li k-putevi skracuju puteve izmedju cvora i i j
  for (int k = 0; k < brojCvorova; k++)
    for (int i = 0; i < brojCvorova; i++)
      for (int j = 0; j < brojCvorova; j++)
        if (najkraciPut[i][k] != INF && najkraciPut[k][j] != INF
            && najkraciPut[i][k] + najkraciPut[k][j] < najkraciPut[i][j]){
          najkraciPut[i][j] = najkraciPut[i][k] + najkraciPut[k][j];
          // postavljamo da je na putu od cvora i do cvora j
          // maksimalna oznaka cvora jednaka k
```

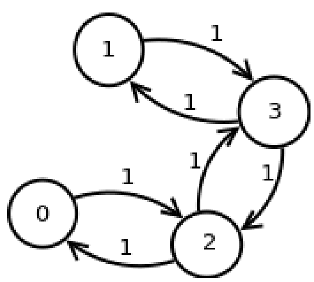
```
          put[i][j] = k;
        }

  // vrsimo proveru da li je u grafu postojao ciklus negativne duzine
  bool negativniCiklus = false;
  for (i = 0; i < brojCvorova; i++)
    if (najkraciPut[i][i] < 0){
      cout << "U grafu postoji ciklus negativne duzine" << endl;
      negativniCiklus = true;
      break;
    }
  // ako ne postoji ciklus negativne duzine
  // stampamo sve najkrace puteve
  if (!negativniCiklus)
    odstampajPuteve(najkraciPut, put, brojCvorova);
}

int main() {

  // broj cvorova u grafu
  int n;
  cin >> n;
  // matrica susedstva grafa koji sadrzi n cvorova
  vector<vector<int>> M(n);
  // ucitavamo matricu susedstva tezinskog grafa
  for (int i = 0; i < n; i++) {
    M[i].resize(n);
    for (int j = 0; j < n; j++){
      cin >> M[i][j];
      // ako je uneto -1, tezinu grane postavljamo na +beskonacno
      // pretpostavljamo da u grafu ne postoji grana negativne tezine
      if (M[i][j] == -1)
        M[i][j] = INF;
    }
  }
  // racunamo najkrace puteve izmedju svaka dva cvora
  sviNajkraciPutevi(M);
  return 0;
}
```

Napomenimo da je neophodno da spoljašnja petlja kontroliše parametar k koji ograničava tip dozvoljenih puteva, dok se unutrašnje dve petlje koriste se za proveru svih parova čvorova. Zapaža se da se ova provera može izvršavati sa parovima čvorova u proizvoljnom redosledu, jer je svaka od provera potpuno nezavisna od ostalih. Takođe, važno je primetiti da Flojd-Varšalov algoritam radi korektno i za grafove koji imaju negativne težine grana (sve dok ne postoji ciklus negativne težine). To je posledica toga da korektnost algoritma ne zavisi od toga da su težine grana u grafu nenegativne.

Razmotrimo primer grafa sa slike 3 kod koga su težine svih grana jednake 1. Neka su redni brojevi čvorova 0, 1, 2 i 3 sa slike redom jednaki 1, 2, 3 i 4. U njemu postoji put od čvora 0 do čvora 1 dužine 3 i on predstavlja najkraći put od čvora 0 do čvora 1. Razmotrimo naredni redosled petlji:

```
for (int i = 0; i < brojCvorova; i++)
  for (int j = 0; j < brojCvorova; j++)
    for (int k = 0; k < brojCvorova; k++)
      if (najkraciPut[i][k] + najkraciPut[k][j] < najkraciPut[i][j]){
        najkraciPut[i][j] = najkraciPut[i][k] + najkraciPut[k][j];
```

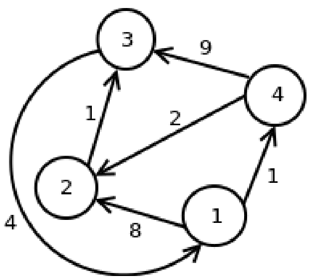
pri čemu promenljiva k kontroliše tip dozvoljenih puteva, i polazni čvor, a j krajnji čvor puta. Ovaj algoritam odgovara tome da se za svaka dva fiksirana čvora numerisana vrednostima i i j prolazi kroz sve čvorove k i razmatra da li postoji put preko čvora k. Za vrednost promenljivih $i = 0$ i $j = 1$ proveravale bi se redom vrednosti $0, 1, 2$ i 3 za k i pošto ne postoji k tako da istovremeno postoji i grana (i, m) i grana (m, j), put od čvora 0 do čvora 1 ne bi bio pronađen, iako on postoji u grafu. Naime, da bismo otkrili najkraći put (tj. 4-put) od čvora 0 do čvora 1 potrebno je prethodno odrediti najkraći 3-put od čvora 0 do čvora 1, što u ovoj varijanti algoritma nije slučaj.



Slika 3: Usmereni težinski graf kod koga modifikacija Flojd-Varšalovog algoritma kod koga unutrašnja petlja kontroliše tip dozvoljenih puteva ne vraća ispravni rezultat.

Ukoliko slučajno polazni graf ima ciklus negativne težine, to se može utvrditi tako što će nakon izvršavanja Flojd-Varšalovog algoritma dužina najkraćeg puta od nekog čvora do njega samog biti manja od 0, odnosno neka vrednost na dijagonali matrice `najkraciPut` biće manja od 0. Ovo važi jer je inicijalno `najkraciPut[i][i] = 0` za svako i. Put od čvora i do njega samog može biti ažuriran samo ako ima dužinu manju od 0, tj. ako postoji put negativne težine od čvora i do njega samog.

Složenost: Za svaku vrednost za k algoritam izvršava jedno sabiranje i jedno upoređivanje za svaki par čvorova. Broj koraka indukcije je $|V|$, pa je ukupan broj sabiranja, odnosno upoređivanja, najviše $|V|^3$. Prisetimo se da je vremenska složenost Dajkstrinog algoritma za nalaženje najkraćih puteva od jednog zadatog čvora u grafu koji ne sadrži grane negativne dužine $O((|V| + |E|) \log |V|)$. Ako je graf gust, pa je broj grana $\Theta(|V|^2)$, onda je Flojd-Varšalov algoritam efikasniji od izvršavanja za svaki čvor algoritma za najkraće puteve od datog čvora. S druge strane, ako graf nije gust (pa ima na primer $O(|V|)$ grana), onda je bolja vremenska složenost $O(|V|(|V| + |E|) \log |V|)$ koja potiče od $|V|$ puta upotrebljenog algoritma za najkraće puteve od jednog čvora. Jedna od prednosti Flojd-Varšalovog algoritma je, svakako, i njegova jednostavna realizacija.



Slika 4: Usmereni težinski graf za koji tražimo najkraći put između svaka dva čvora.

Razmotrimo postupak određivanja najkraćih puteva između svaka dva čvora u grafu sa slike 4 – on bi se sastojao iz narednih koraka:

$$k=0\text{: }\texttt{najkraciPut}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{pmatrix} \end{array} \qquad \texttt{put}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 1 & -1 & 1 \\ -1 & 0 & 2 & -1 \\ 3 & -1 & 0 & -1 \\ -1 & 4 & 4 & 0 \end{pmatrix} \end{array}$$

$$k=1\text{: }\texttt{najkraciPut}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{pmatrix} \end{array} \qquad \texttt{put}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 1 & -1 & 1 \\ -1 & 0 & 2 & -1 \\ 3 & 1 & 0 & 1 \\ -1 & 4 & 4 & 0 \end{pmatrix} \end{array}$$

$$k=2\text{: }\texttt{najkraciPut}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{pmatrix} \end{array} \qquad \texttt{put}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 1 & 2 & 1 \\ -1 & 0 & 2 & -1 \\ 3 & 1 & 0 & 1 \\ -1 & 4 & 2 & 0 \end{pmatrix} \end{array}$$

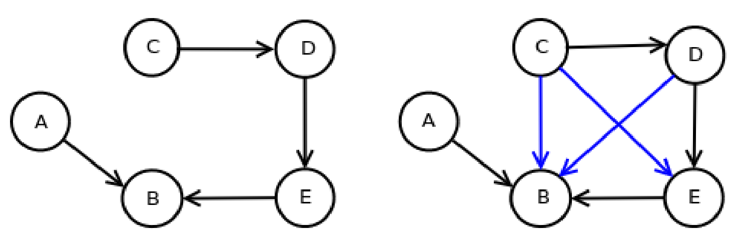
$$k=3\text{: }\texttt{najkraciPut}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{pmatrix} \end{array} \qquad \texttt{put}\text{: } \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 1 & 2 & 1 \\ 3 & 0 & 2 & 3 \\ 3 & 1 & 0 & 1 \\ 3 & 4 & 2 & 0 \end{pmatrix} \end{array}$$

$$k=4\text{: }\texttt{najkraciPut: }\begin{array}{c c} & \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{pmatrix}0&3&4&1\\5&0&1&6\\4&7&0&5\\7&2&3&0\end{pmatrix}\end{array} \qquad \texttt{put: }\begin{array}{c c} & \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{pmatrix}0&4&4&1\\3&0&2&3\\3&4&0&1\\3&4&2&0\end{pmatrix}\end{array}$$

Iz poslednje matrice `najkraciPut` možemo pročitati da je najkraći put od čvora 1 do čvora 3 dužine 4, a iz matrice `put` da je maksimalni indeks čvora na tom najkraćem putu jednak 4. Dakle, da bismo rekonstruisali najkraći put, potrebno je da na najkraći put od čvora 1 do čvora 4 nadovežemo najkraći put od čvora 4 do čvora 3. Iz matrice `put` možemo pročitati da je najveći indeks čvora na najkraćem putu od 1 do 4 baš jednak 1 što nam govori da je taj put direktna grana između tih čvorova. Iz matrice `put` možemo takođe pročitati da je najveći indeks čvora na najkraćem putu od 4 do 3 jednak 2 što nam govori da taj put određujemo tako što na najkraći put od čvora 4 do čvora 2 nadovežemo najkraći put od čvora 2 do čvora 3. Iz matrice `put` možemo zaključiti da oba ova puta odgovaraju direktnim granama. Konačno, zaključujemo da je najkraći put od čvora 1 do čvora 3 jednak $1, 4, 2, 3$.

Tranzitivno zatvorenje grafa

Za zadati usmereni graf $G = (V, E)$ njegovo *tranzitivno zatvorenje* (eng. transitive closure) $C = (V, F)$ je usmereni graf u kome grana (u, w) između čvorova u i w postoji ako i samo ako u grafu G postoji usmereni put od čvora u do čvora w. Na slici 5 prikazan je jedan usmeren graf i njegovo tranzitivno zatvorenje.



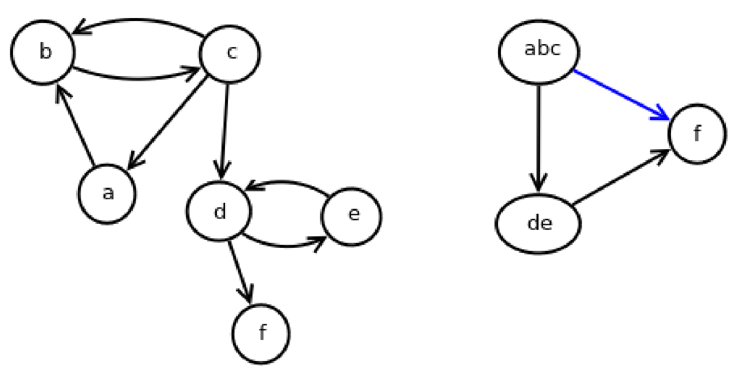
Slika 5: Graf i njegovo tranzitivno zatvorenje: plavom bojom istaknute su grane koje su dodate u polazni graf.

Postoji mnogo primena tranzitivnog zatvorenja, pa je važno imati efikasan algoritam za njegovo nalaženje. Na primer, tabelu u programu za tabelarna izračunavanja (npr. Microsoft Excel) možemo predstaviti u vidu usmerenog grafa: ćelije tabele odgovaraju čvorovima, a grana od čvora a do čvora b postoji ako vrednost koja se računa u ćeliji b zavisi od vrednosti ćelije a. Kada se izmeni neka od vrednosti u tabeli, potrebno je ažurirati vrednosti svih ćelija koje od nje zavise, odnosno svih čvorova koji su dostižni iz date ćelije. Te ćelije se mogu utvrditi određivanjem tranzitivnog zatvorenja datog grafa. Takođe, ako neki skup aerodroma razmotrimo kao skup čvorova, a informaciju da postoji direktan let od jednog do drugog aerodroma predstavimo usmerenom granom između odgovarajućih čvorova, onda nam tranzitivno zatvorenje grafa daje informaciju o tome sa kog aerodroma je moguće stići do kog direktnim letom ili putem više povezanih letova.

Problem: Odrediti tranzitivno zatvorenje zadatog usmerenog grafa $G = (V, E)$.

Postoji više načina za računanje tranzitivnog zatvorenja datog grafa. Na primer, može se iz svakog čvora pokrenuti pretraga u dubinu ili širinu i sačuvati informacija o svim dostižnim čvorovima. Ovaj algoritam bio bi vremenske složenosti $O(|V|(|V|+|E|))$ i ima dobre performanse ako je graf redak, dok za guste grafove postaje složenosti $O(|V|^3)$.

Ako je tranzitivno zatvorenje grafa G gust graf, bolji pristup je prvo izračunati komponente jake povezanosti grafa G algoritmom linearne vremenske složenosti. Za svaka dva čvora iz iste komponente jake povezanosti važi da su međusobno dostižna, a ako postoji grana (u, v) koja povezuje čvorove iz različitih jakih komponenti povezanosti, svaki čvor iz komponente kojoj pripada čvor v je dostižan iz svakog čvora komponente kojoj pripada čvor u. Dakle, problem se svodi na pronalaženje tranzitivnog zatvorenja komprimovanog grafa, sačinjenog od jakih komponenti povezanosti, koji obično ima dosta manje čvorova i grana (slika 6).

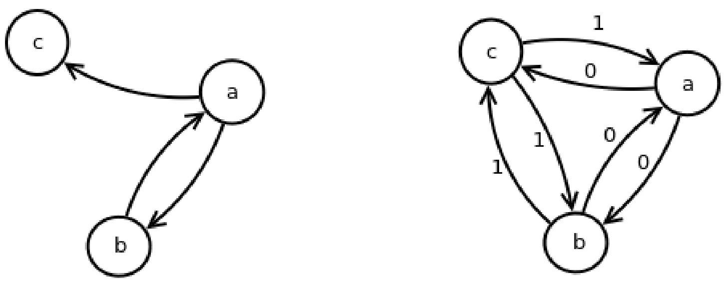


Slika 6: Graf i tranzitivno zatvorenje njegovog komprimovanog grafa (plavom bojom označene su grane koje su dodate u graf).

Treći način da rešimo ovaj problem jeste redukcijom (svođenjem) na drugi problem. Pokažimo kako se proizvoljni ulaz za problem računanja tranzitivnog zatvorenja grafa može svesti na ulaz za drugi problem, koji umemo da rešimo; nakon toga rešenje drugog problema transformišemo u rešenje problema tranzitivnog zatvorenja. Problem na koji vršimo svođenje je problem određivanja svih najkraćih puteva u grafu.

Dakle, neka je zadatak odrediti tranzitivno zatvorenje grafa $G = (V, E)$ i neka je $G' = (V, E')$ kompletni usmereni težinski graf (graf kod koga za svaki par čvorova postoje obe grane, u oba smera) sa istim skupom čvorova kao graf G.

Grani $e \in E'$ dodeljuje se težina 0 ako $e \in E$, odnosno 1 u protivnom (slika 7). Sada za graf G' rešavamo problem nalaženja svih najkraćih puteva. U grafu G postoji put između čvorova v i w ako i samo ako je dužina najkraćeg puta od v do w u grafu G' jednaka 0. Dokažimo ovo tvrđenje: pretpostavimo da u grafu G postoji put između čvorova v i w. Težine svih grana na tom putu u grafu G' jednake su 0 i pošto su u grafu G' sve grane težine 0 ili 1, najkraći mogući put između dva čvora jednak je 0 (u situaciji kada su težine svih grana na tom putu jednake 0) i ovaj put biće pronađen kao najkraći put između čvorova v i w u grafu G'. Dokažimo suprotni smer tvrđenja. Pretpostavimo da je dužina najkraćeg puta između čvorova v i w u grafu G' jednaka 0. Odatle sledi da je težina svake od grana na tom putu jednaka 0, te svaka od njih postoji i u grafu G. Dakle, u grafu G postoji put između čvorova v i w.



Slika 7: Usmereni graf G i usmereni težinski graf G' potreban za izvođenje redukcije sa problema tranzitivnog zatvorenja grafa na problem računanja svih najkraćih puteva.

Drugim rečima, rešenje problema svih najkraćih puteva neposredno se transformiše u rešenje problema tranzitivnog zatvorenja grafa.

Umesto da direktno primenimo algoritam za određivanje svih najkraćih puteva u grafu, možemo ga vremenski i prostorno optimizovati tako da direktno rešava problem tranzitivnog zatvorenja grafa. Primetimo da su dužine grana u grafu G' jednake 0 ili 1 i da nas jedino interesuje da li je dužina najkraćeg puta u grafu G' jednaka 0 ili je strogo veća od nule (a u slučaju kada je veća od nule ne interesuje nas njena konkretna vrednost, tj. da li je ona jednaka 10, 7 ili 2). Umesto da koristimo celobrojnu matricu u kojoj će se pamtiti dužine najkraćih puteva između svaka dva čvora, možemo koristiti logičku matricu koja će na poziciji (i, j) sadržati vrednost `true` ako je čvor j dostižan iz čvora i, a inače `false`[1]. Takođe, umesto da koristimo aritmetičke operacije, možemo preći na logičke operacije: umesto operacije sabiranja koristimo logičku konjunkciju.

```
// funkcija koja za svaka dva cvora utvrdjuje
// da li izmedju njih postoji put
void izracunajTranzitivnoZatvorenje(
```

[1] Primetimo da je dosadašnja vrednost 0 kodirala postojanje puta u grafu G, a logička vrednost 0 (tj. vrednost ‘false‘) sad kodira nepostojanje puta u grafu G.

```
    vector<vector<bool>> matricaPovezanosti) {

  // inicijalizujemo matricu tranzitivnog zatvorenja
  // na matricu povezanosti grafa
  vector<vector<bool>> tranzitivnoZatvorenje = matricaPovezanosti;
  int brojCvorova = matricaPovezanosti.size();

  // proveravamo da li postoji put kroz cvor sa oznakom k
  for (int k = 0; k < brojCvorova; k++)
    for (int i = 0; i < brojCvorova; i++)
      for (int j = 0; j < brojCvorova; j++)
        if (tranzitivnoZatvorenje[i][k] && tranzitivnoZatvorenje[k][j])
          tranzitivnoZatvorenje[i][j] = true;

  cout << "Matrica susedstva grafa koji"
       << " predstavlja tranzitivno zatvorenje je" << endl;
  for (int i = 0; i < brojCvorova; i++){
    for (int j = 0; j < brojCvorova; j++)
      cout << tranzitivnoZatvorenje[i][j] << " ";
    cout << endl;
  }
}

int main() {
  // broj cvorova u grafu
  int n;
  cin >> n;
  // matrica susedstva grafa koji sadrzi n cvorova
  vector<vector<bool>> M(n);
  // ucitavamo matricu susedstva
  for (int i = 0; i < n; i++) {
    M[i].resize(n);
    for (int j = 0; j < n; j++){
      // realizujemo ucitavanje logickih vrednosti
      int x;
      cin >> x;
      M[i][j] = x == 1;
    }
  }

  izracunajTranzitivnoZatvorenje(M);
  return 0;
}
```

Matrica `tranzitivnoZatvorenje` na kraju izvršavanja algoritma kodira informacije o dostižnosti u polaznom grafu G, odnosno predstavljaće skup grana u tranzitivnom zatvorenju grafa G.

Činjenica da možemo da svedemo jedan problem na drugi znači da je prvi problem opštiji od drugog, odnosno da je drugi problem specijalni slučaj prvog. Obično su opštija rešenja skuplja, složenija. Posle korišćenja redukcije preporučljivo je pokušati sa popravkom dobijenog rešenja, koristeći specijalne osobine problema.

Razmotrimo osnovni korak algoritma, naredbu `if`. Ona se sastoji od dve provere: da li važi `tranzitivnoZatvorenje[i,m]` i da li važi `tranzitivnoZatvorenje[m,j]`. Akcija se preduzima samo ako su oba uslova ispunjena. Ova `if` naredba izvršava se *brojCvorova* puta za svaki par čvorova. Svaka popravka ove naredbe vodila bi bitnoj popravci algoritma. Moraju li se svaki put proveravati oba uslova? Prva provera zavisi samo od vrednosti i i m, a druga zavisi samo od vrednosti m i j. Zbog toga se prva provera može za fiksirane vrednosti i i m izvršiti samo jednom (umesto *brojCvorova* puta):

- ako prvi uslov nije ispunjen, onda se drugi ne mora proveravati ni za jednu vrednost j,
- ako je pak prvi uslov ispunjen, onda se njegova ispunjenost ne mora ponovo proveravati za svaku vrednost j.

Ova promena je ugrađena u poboljšani algoritam čiji je ključni fragment dat u nastavku. Asimptotska složenost ostaje nepromenjena, ali se algoritam u proseku izvršava dva puta brže.

```
// proveravamo da li postoji put kroz cvor sa oznakom m
for (int k = 0; k < brojCvorova; k++)
  for (int i = 0; i < brojCvorova; i++)
    if (tranzitivnoZatvorenje[i][k])
      for (int j = 0; j < brojCvorova; j++)
        if (tranzitivnoZatvorenje[k][j])
          tranzitivnoZatvorenje[i][j] = true;
```

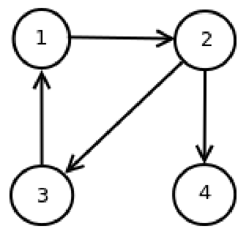
Ovaj algoritam može se dalje usavršiti. Linija

```
if (tranzitivnoZatvorenje[k][j])
   tranzitivnoZatvorenje[i][j] = true;
```

može se ekvivalentno zameniti linijom

```
tranzitivnoZatvorenje[i][j] =
    tranzitivnoZatvorenje[i][j] | tranzitivnoZatvorenje[k][j];
```

Zapaža se da se posle ove zamene u unutrašnjoj petlji algoritma operacija `or` primenjuje na vrstu i i vrstu k matrice *tranzitivnoZatvorenje*, a rezultat je nova vrsta i. Zbog toga, ako je $n = brojCvorova \leq 64$, vrste matrice *tranzitivnoZatvorenje* predstavljaju niz nula i jedinica i mogu se tumačiti kao n bitova u binarnoj reprezentacija celog broja, pa se primena operacije `or` na vrste može zameniti bitskom `or` operacijom dva cela broja, što je n puta brže. Ako je $n > 64$, onda se vrsta može predstaviti nizom 64-bitnih celih brojeva, pa se algoritam izvršava približno 64 puta brže. Asimptotska složenost je i dalje $O(n^3)$, ali ubrzanje za faktor 64 nije zanemarljivo.



Slika 8: Primer grafa za koji je potrebno odrediti tranzitivno zatvorenje.

Razmotrimo primer izračunavanja tranzitivnog zatvorenja grafa sa slike 8. On bi se sastojao iz narednih koraka:

$$k=0:\ \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1\\2\\3\\4 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad k=1:\ \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1\\2\\3\\4 \end{array} & \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$k=2:\ \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1\\2\\3\\4 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad k=3:\ \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1\\2\\3\\4 \end{array} & \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$k=4:\ \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1\\2\\3\\4 \end{array} & \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Primetimo da u grafu sa slike 8 ne postoji direktna grana od čvora 3 do čvora 2, ali postoji put dužine dva preko čvora 1. Slično, od čvora 3 do čvora 4 ne postoji direktna grana, ali postoji put dužine 3, preko čvorova 1 i 2.

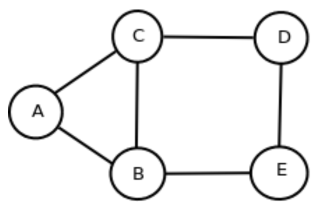
Čitaocima se ostavlja za razmišljanje pitanje kako bi izgledalo tranzitivno zatvorenje neusmerenog grafa.

Interesantan problem u vezi sa nalaženjem tranzitivnog zatvorenja je i problem određivanja *tranzitivne redukcije* grafa (eng. transitive reduction). Ova operacija predstavlja operaciju inverznu operaciji pronalaženja tranzitivnog zatvorenja grafa: cilj je da se za dati usmereni graf konstruiše usmereni graf sa istim skupom čvorova i što manjim skupom grana, a da se pritom ne promeni relacija dostižnosti. Tranzitivno zatvorenje grafa G jednako je tranzitivnom zatvorenju tranzitivne redukcije grafa G. Iako je tranzitivno zatvorenje grafa G jedinstveno određeno, u opštem slučaju graf može imati više različitih tranzitivnih redukcija.

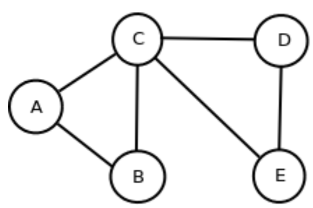
Putevi i ciklusi u grafovima

Ojlerovi putevi i ciklusi

Ojlerov put (eng. Eulerian trail, Eulerian path) je put u grafu koji prolazi kroz svaku granu grafa tačno jednom. Na primer, u neusmerenom grafu prikazanom na slici 9 postoji Ojlerov put C, D, E, B, C, A, B. *Ojlerov ciklus* (eng. Eulerian circuit, Eulerian cycle) je Ojlerov put čiji se početni i krajnji čvor poklapaju. U grafu sa slike 9 ne postoji Ojlerov ciklus, dok u grafu prikazanom na slici 10 postoji Ojlerov ciklus C, D, E, C, A, B, C.

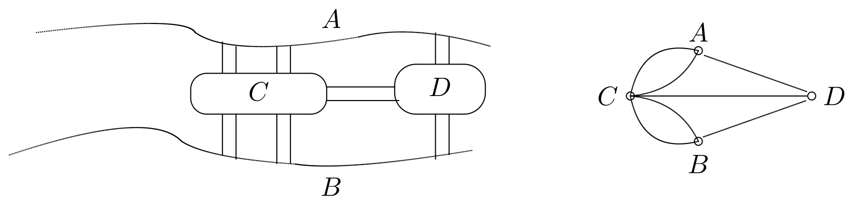


Slika 9: Neusmeren graf koji sadrži Ojlerov put, ali ne sadrži Ojlerov ciklus.



Slika 10: Neusmeren graf koji sadrži Ojlerov ciklus.

Pojam Ojlerovih grafova u vezi je sa, kako se smatra, prvim rešenim problemom teorije grafova. Švajcarski matematičar Leonard Ojler naišao je 1736. godine na sledeći zadatak. Grad Kenigsberg, danas Kalinjingrad, leži na obalama i na dva ostrva na reci Pregel, kao što je prikazano na slici 11. Grad je povezan sa sedam mostova. Pitanje koje je mučilo mnoge tadašnje građane Kenigsberga bilo je da li je moguće početi šetnju iz bilo koje tačke u gradu i vratiti se u polaznu tačku, prelazeći pri tome svaki most tačno jednom. Ovaj problem se može formulisati kao sledeći problem iz teorije grafova: da li je moguće u neusmerenom povezanom grafu pronaći ciklus, koji svaku granu sadrži tačno jednom, tzv. Ojlerov ciklus. Odnosno, da li je moguće nacrtati graf sa slike 11 ne dižući olovku sa papira, tako da olovka svoj put završi na mestu sa koga je i krenula. Napomenimo da ovaj graf ima višestruke grane između parova čvorova, pa strogo gledano po definiciji nije graf, već multigraf. Ojler je rešio problem, dokazavši da je ovakav obilazak moguć ako i samo ako je graf povezan i svi njegovi čvorovi imaju paran stepen. Grafovi koji sadrže Ojlerov ciklus zovu se *Ojlerovi grafovi*. Pošto “graf” na slici 11 ima čvorove neparnog stepena, zaključujemo da problem Kenigsberških mostova nema rešenje.



Slika 11: Problem Kenigsberških mostova, i odgovarajući multigraf.

Teorema: U neusmerenom grafu $G = (V, E)$ postoji Ojlerov ciklus ako i samo ako je G povezan graf u kome svi čvorovi imaju parni stepen.

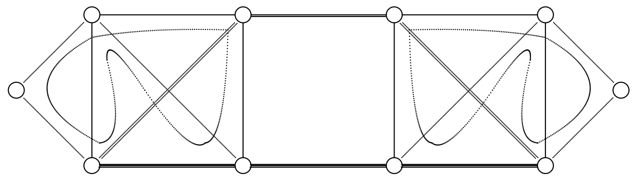
Dokaz: Lako je pokazati da ako u grafu postoji Ojlerov ciklus, onda svi čvorovi grafa moraju imati paran stepen. Naime, za vreme obilaska ciklusa, u svaki čvor se ulazi isto toliko puta koliko puta se iz njega izlazi. Pošto se svaka grana prolazi tačno jednom, broj grana susednih proizvoljnom čvoru mora biti paran.

Da bismo indukcijom dokazali da je ovaj uslov i dovoljan da bi graf imao Ojlerov ciklus, moramo najpre da izaberemo parametar po kome će biti izvedena indukcija. Taj izbor treba da omogući smanjivanje problema, bez njegove promene. Ako uklonimo čvor ili granu iz grafa, stepeni čvorova u dobijenom grafu nisu više svi parni. Trebalo bi da uklonimo takav skup grana S, tako da za svaki čvor v grafa G broj grana iz skupa S susednih sa v bude paran (makar i 0). Proizvoljan ciklus zadovoljava ovaj uslov, pa se postavlja pitanje da li Ojlerov graf uvek sadrži ciklus. Pretpostavimo da smo započeli obilazak grafa iz proizvoljnog čvora v proizvoljnim redosledom. Sigurno je da će se tokom obilaska ranije ili kasnije naići na već obiđeni čvor, jer kad god uđemo u neki čvor, smanjujemo njegov stepen za jedan, činimo ga neparnim, pa ga uvek možemo i napustiti. Naravno, ovakav obilazak ne mora da sadrži sve grane grafa. Dakle važi da u svakom Ojlerovom grafu mora da postoji neki ciklus.

Sada možemo da formulišemo induktivnu hipotezu i dokažemo teoremu.

Induktivna hipoteza: Povezani neusmereni graf sa $< m$ grana čiji svi čvorovi imaju paran stepen, sadrži Ojlerov ciklus, koji se može efektivno pronaći.

Posmatrajmo graf $G = (V, E)$ koji sadrži m grana. Neka je P neki ciklus u grafu G, i neka je G' graf dobijen uklanjanjem grana ciklusa P iz grafa G. Stepeni svih čvorova u grafu G' su parni, jer je broj uklonjenih grana susednih bilo kom čvoru paran. Ipak se induktivna hipoteza ne može primeniti na graf G', jer on ne mora biti povezan (slika 12). Neka su $G'_1, G'_2, \ldots, G'_k$ komponente povezanosti grafa G'. U svakoj komponenti stepeni svih čvorova su parni. Pored toga, broj grana u svakoj komponenti je manji od m jer je njihov ukupan broj grana manji je od m. Prema tome, induktivna hipoteza se može primeniti na sve komponente: u



Slika 12: Primer konstrukcije Ojlerovog ciklusa indukcijom. Punom linijom izvučene su grane pomoćnog ciklusa. Izbacivanjem grana ovog ciklusa iz grafa, dobija se graf sa dve komponente povezanosti.

svakoj komponenti G'_i postoji Ojlerov ciklus P'_i, i mi znamo da ga pronađemo. Potrebno je sada sve ove cikluse objediniti sa pomoćnim ciklusom P u jedan Ojlerov ciklus za graf G. Polazimo iz bilo kog čvora ciklusa P ("magistralnog puta") sve dok ne dođemo do nekog čvora v_j koji pripada nekoj komponenti G'_j. Tada obilazimo komponentu G'_j ciklusom P'_j ("lokalnim putem") i vraćamo se u čvor v_j. Nastavljajući na taj način, obilazeći cikluse komponenti u trenutku nailaska na njih, na kraju ćemo se vratiti u polazni čvor. U tom trenutku sve grane grafa G smo prošli tačno jednom, što znači da je konstruisan Ojlerov ciklus. □

Ovaj dokaz daje i efikasan rekurzivni algoritam za konstrukciju Ojlerovog ciklusa u grafu.

Videli smo da graf može imati Ojlerov put, a da istovremeno nema Ojlerov ciklus (slika 9). Što se Ojlerovih puteva tiče, važi da neusmereni graf ima Ojlerov put ako i samo ako je povezan i važi jedan od narednih uslova:

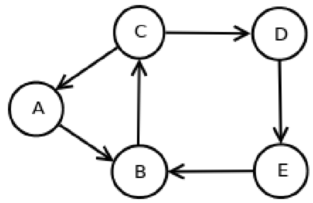
- stepen svakog čvora je paran ili
- stepen tačno dva čvora je neparan, a ostalih čvorova je paran.

Ako važi prva stavka onda je svaki Ojlerov put istovremeno i Ojlerov ciklus. Ukoliko važi druga stavka, čvorovi neparnog stepena su početni i krajnji čvor Ojlerovog puta i u ovakvom grafu ne postoji Ojlerov ciklus. U grafu sa slike 9 čvorovi B i C su neparnog stepena, dok su ostali čvorovi parnog stepena, te čvorovi B i C predstavljaju početni i krajnji čvor Ojlerovog puta. Dodatno, u ovom grafu ne postoji Ojlerov ciklus.

U usmerenom grafu postoji Ojlerov put ako i samo ako je graf jako povezan i važi jedan od narednih uslova:

- ulazni i izlazni stepeni svih čvorova su međusobno jednaki ili
- ulazni stepen veći je za jedan od izlaznog stepena tačno jednog čvora, izlazni stepen veći je za jedan od ulaznog stepena tačno jednog čvora, dok su za sve ostale čvorove ulazni i izlazni stepen jednaki.

U prvom slučaju, svaki Ojlerov put je i Ojlerov ciklus, a u drugom slučaju Ojlerov put počinje u čvoru čiji je izlazni stepen veći za jedan, a završava se u čvoru čiji je ulazni stepen veći za jedan.



Slika 13: Usmereni graf koji sadrži Ojlerov put, ali ne sadrži Ojlerov ciklus.

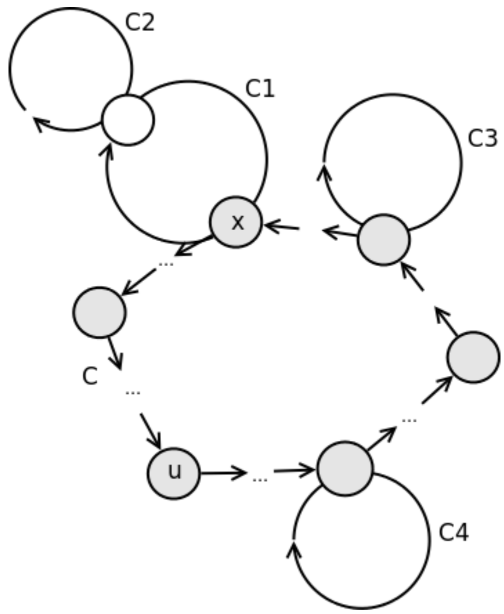
U grafu sa slike 13 možemo videti da za čvorove A, E i D važi da im je isti ulazni i izlazni stepen, ulazni stepen čvora B je veći za jedan od izlaznog, dok za čvor C važi da je ulazni stepen za jedan manji od izlaznog stepena. U njemu postoji Ojlerov put koji počinje u čvoru C, a završava se u čvoru B, dok Ojlerov ciklus ne postoji.

Jedan efikasan način za konstruisanje Ojlerovog ciklusa u Ojlerovom grafu predstavlja *Hirholcerov algoritam*, zasnovan na ideji prikazanoj u dokazu prethodne teoreme. Algoritam se može primeniti i na neusmerene i na usmerene grafove. On se sastoji od nekoliko etapa, pri čemu se u svakoj etapi prethodno pronađeni ciklus proširuje narednim ciklusom u novi, veći ciklus. Postupak se zaustavlja kada se sve grane grafa dodaju u ciklus.

Algoritam kreće iz proizvoljnog čvora u u grafu i proizvoljnom neposećenom granom odlazi do njegovog suseda. Ovaj korak se ponavlja sve dok se ne vratimo u polazni čvor u. U njega se moramo vratiti u nekom momentu jer je stepen svakog čvora paran. Na ovaj način konstruiše se inicijalni ciklus. Ukoliko on prolazi skupom svih grana, Ojlerov ciklus je konstruisan. Inače, ciklus se proširuje na sledeći način: polazeći iz čvora u duž ciklusa pronalazi se čvor x koji pripada tekućem ciklusu i koji ima granu koja nije uključena u ciklus (u slučaju usmerenog grafa vraćamo se unazad duž ciklusa i tražimo čvor koji ima izlaznu granu koja nije uključena u ciklus). Tom granom se kreće u otkrivanje novog ciklusa koji se sastoji isključivo od grana koje još uvek nisu u prethodnom ciklusu. Put će se pre ili kasnije vratiti u čvor x i time će biti otkriven novi ciklus koji dodajemo u tekući ciklus. Na ovaj način se od ova dva ciklusa konstruiše novi ciklus. Postupak se nastavlja dok pronađeni ciklus ne obuhvati sve grane grafa.

Razmotrimo primer sa slike 14. Obilazak kreće iz čvora u i najpre bismo odredili ciklus C čiji su čvorovi označeni sivom bojom. Nakon toga bismo se unazad vraćali čvorovima ciklusa C počev od čvora u i redom bismo objedinjavali cikluse C_1, C_2, C_3 i C_4 sa glavnim ciklusom.

Čvorove prvog ciklusa stavljamo na stek u redosledu obilaska. Nakon toga iz steka uklanjamo jedan po jedan čvor počev od čvora u, prebacujemo ga u Ojlerov ciklus koji konstruišemo i razmatramo da li postoji neka grana iz tog čvora koja do sada nije posećena. Ako postoji, iz tog čvora startujemo obilazak novog ciklusa i sve čvorove koje smo prošli stavljamo na stek. Kada stek postane



Slika 14: Objedinjavanje ciklusa u novi ciklus.

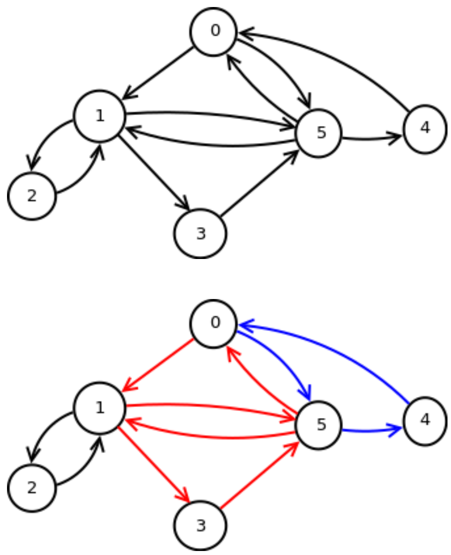
prazan, to znači da smo za sve čvorove razmotrili sve grane koje polaze iz njih. Umesto da pamtimo informaciju o posećenosti čvorova, možemo uklanjati grane iz grafa.

Ako graf G sadrži samo Ojlerov put, a ne i Ojlerov ciklus, u graf G se može dodati grana kojom se postiže uslov da graf ima Ojlerov ciklus i zatim iskoristiti prethodni algoritam. Nakon pronalaska Ojlerovog ciklusa u dopunjenom grafu ta grana se uklanja iz ciklusa (i grafa).

```
vector<vector<int>> listaSuseda {{1}, {2, 3}, {1, 3}, {0, 2, 4},
                                  {5}, {3}};

void ispisiOjlerovCiklus(){

  // stek na koji stavljamo cvorove u redosledu obilaska
  stack<int> tekuciPut;
  // ciklus koji u etapama nadogradjujemo
  vector<int> ojlerovCiklus;

  // na stek dodajemo polazni cvor
  tekuciPut.push(0);
  int tekuciCvor = 0;

  // sve dok postoji neki cvor na tekucem putu
```

```
  while (!tekuciPut.empty()){
    // ako iz tekuceg cvora postoji jos neka grana koju nismo posetili
    if (listaSuseda[tekuciCvor].size()){
      // tekuci cvor dodajemo u tekuci put
      tekuciPut.push(tekuciCvor);

      // pronalazimo cvor do koga postoji grana iz tekuceg cvora;
      // uzimamo poslednji iz liste povezanosti jer je njega lako
      // ukloniti iz liste povezanosti
      int naredniCvor = listaSuseda[tekuciCvor].back();
      // brisemo granu iz tekuceg ka narednom cvoru
      listaSuseda[tekuciCvor].pop_back();

      // napredujemo ka narednom cvoru
      tekuciCvor = naredniCvor;
    }
    // ako iz tog cvora ne postoji neposecena grana
    else{
      // vracamo se unazad da bismo nasli preostale cikluse
      ojlerovCiklus.push_back(tekuciCvor);
      tekuciCvor = tekuciPut.top();
      tekuciPut.pop();
    }
  }

  // stampamo pronadjeni Ojlerov ciklus u suprotnom redosledu cvorova
  cout << "Ojlerov ciklus u datom grafu je: ";
  for (int i = ojlerovCiklus.size() - 1; i >= 0; i--){
    cout << ojlerovCiklus[i];
    if (i)
      cout << " - ";
  }
  cout << endl;
}

int main() {
  ispisiOjlerovCiklus();
  return 0;
}
```

Na slici 15 dat je primer usmerenog Ojlerovog grafa. Izvršavanje Hirholcerovog algoritma bi krenulo od čvora 0 i pronašlo bi recimo ciklus $0, 5, 4, 0$. Grane ovog ciklusa se uklanjaju iz grafa. Prilikom povratka u čvor 0 proverava se da li postoji još neka grana u grafu koja polazi iz čvora 0 i pošto postoji, krenulo bi se granom ka čvoru 1 i pronašao novi ciklus $0, 1, 5, 1, 3, 5, 0$. Tekući put kojim se krećemo sadrži čvorove $0, 5, 4, 0, 1, 5, 1, 3, 5, 0$. U ovom trenutku konstatujemo



Slika 15: Ilustracija izvršavanja Hirholcerovog algoritma u slučaju datog usmerenog grafa. Polazni čvor je 0. Stek je na početku prazan, nakon pronalaska prvog ciklusa sadrži čvorove $0, 5, 4, 0$; nakon pronalaska narednog ciklusa sadrži čvorove $0, 5, 4, 0, 1, 5, 1, 3, 5, 0$; nakon pronalaska narednog ciklusa sadrži čvorove $0, 5, 4, 0, 1, 5, 1, 2, 1$. Na kraju stek će biti prazan. Redosled obilaska suseda je u opadajućem redosledu indeksa zbog efikasnosti brisanja elemenata sa kraja vektora.

da iz čvora 0 ne postoji nijedna preostala grana u grafu, te jedan po jedan čvor iz ovog puta prebacujemo s kraja u Ojlerov ciklus; za svaki naredni čvor najpre ispitujemo da li postoji još neka grana iz tog čvora, ako postoji krećemo u potragu za novim ciklusom iz tog čvora, a ako ne postoji onda čvor prebacujemo u Ojlerov ciklus. Nakon prebacivanja čvorova 0, 5 i 3 utvrđuje se da iz čvora 1 postoji grana ka čvoru 2 koja do sada nije bila posećena te otkrivamo novi ciklus $1, 2, 1$ i čvorove ovog ciklusa dodajemo na kraj tekućeg puta. Nakon ovog ciklusa, sve dok ne iscrpemo sve čvorove iz tekućeg puta nećemo pronaći nijednu novu granu. Konačno, zaključujemo da je u ovom grafu Ojlerov ciklus jednak $0, 5, 4, 0, 1, 5, 1, 2, 1, 3, 5, 0$. Sadržaj steka i dela konstruisanog Ojlerovog ciklusa korak po korak prikazani su u tabeli 1.

`tekuciCvor`	`tekuciPut`	`ojlerovCiklus`
0	0	
5	$0, 5$	
4	$0, 5, 4$	
0	$0, 5, 4, 0$	
1	$0, 5, 4, 0, 1$	
5	$0, 5, 4, 0, 1, 5$	
1	$0, 5, 4, 0, 1, 5, 1$	
3	$0, 5, 4, 0, 1, 5, 1, 3$	
5	$0, 5, 4, 0, 1, 5, 1, 3, 5$	
0	$0, 5, 4, 0, 1, 5, 1, 3, 5, 0$	
5	$0, 5, 4, 0, 1, 5, 1, 3, 5$	0
3	$0, 5, 4, 0, 1, 5, 1, 3$	$0, 5$
1	$0, 5, 4, 0, 1, 5, 1$	$0, 5, 3$
2	$0, 5, 4, 0, 1, 5, 1, 2$	$0, 5, 3$
1	$0, 5, 4, 0, 1, 5, 1, 2, 1$	$0, 5, 3$
2	$0, 5, 4, 0, 1, 5, 1, 2$	$0, 5, 3, 1$
1	$0, 5, 4, 0, 1, 5, 1$	$0, 5, 3, 1, 2$
5	$0, 5, 4, 0, 1, 5$	$0, 5, 3, 1, 2, 1$
1	$0, 5, 4, 0, 1$	$0, 5, 3, 1, 2, 1, 5$
0	$0, 5, 4, 0$	$0, 5, 3, 1, 2, 1, 5, 1$
4	$0, 5, 4$	$0, 5, 3, 1, 2, 1, 5, 1, 0$
5	$0, 5$	$0, 5, 3, 1, 2, 1, 5, 1, 0, 4$
0	0	$0, 5, 3, 1, 2, 1, 5, 1, 0, 4, 5$
–		$0, 5, 3, 1, 2, 1, 5, 1, 0, 4, 5, 0$

Tabela 1: Primer izvršavanja Hirholcerovog algoritma za graf sa slike 15

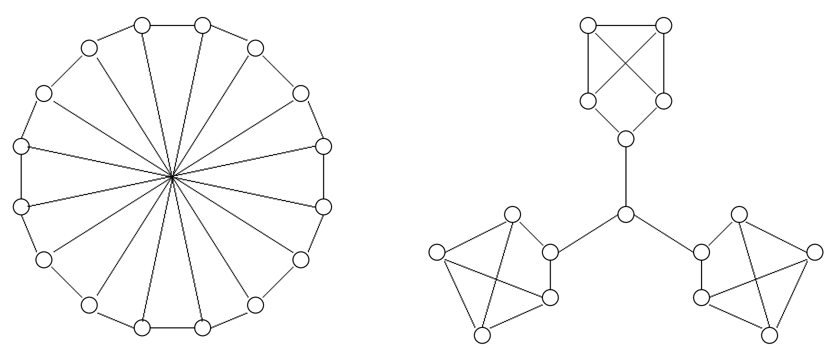
Vreme izvršavanja Hirholcerovog algoritma iznosi $O(|E|)$. Primetimo da ukoliko je graf neusmeren, onda prilikom brisanja neke grane, treba obrisati obe njene kopije: (u, v) i (v, u), što je operacija koja se ne izvršava efikasno u slučaju reprezentacije grafa listama povezanosti.

Pored Hirholcerovog, postoji i Flerijev algoritam za konstrukciju Ojlerovih ciklusa u grafu. On se zasniva na detekciji mostova u grafu i manje je efikasan – njegova složenost iznosi $O(|E|^2)$.

Hamiltonovi putevi i ciklusi

Hamiltonov put (eng. Hamiltonian path) je put koji posećuje svaki čvor grafa tačno jednom. Na primer, graf prikazan na slici 9 sadrži Hamiltonov put A, B, C, D, E. Ako Hamiltonov put počinje i završava se u istom čvoru on se naziva *Hamiltonov ciklus* (eng. Hamiltonian cycle, Hamiltonian circuit). Graf sa slike 9 ima i Hamiltonov ciklus koji počinje i završava se u čvoru A: to je ciklus A, B, E, D, C, A. Graf koji sadrži Hamiltonov ciklus zove se *Hamiltonov graf*. Hamiltonovi ciklusi dobili su ime u čast Vilijama Hamiltona koji je 1857. godine izumeo jednu vrstu slagalice koja uključuje potragu za ovom vrstom ciklusa u grafu sačinjenom od ivica dodekaedra.

Za razliku od problema Ojlerovih ciklusa, problem nalaženja Hamiltonovih ciklusa (odnosno karakterizacije Hamiltonovih grafova) je vrlo težak. Da bi se proverilo da li je graf Ojlerov, dovoljno je znati stepene njegovih čvorova. Za utvrđivanje da li je graf Hamiltonov to nije dovoljno. Zaista, dva grafa prikazana na slici 16 imaju po 16 čvorova stepena 3 (dakle imaju iste stepene čvorova), ali je prvi Hamiltonov, a drugi očigledno nije. Problem ispitivanja da li je dati graf Hamiltonov spada u klasu NP-kompletnih problema i mi se u ovom materijalu nećemo detaljnije baviti njima.



Slika 16: Dva grafa sa po 16 čvorova stepena 3, od kojih je prvi Hamiltonov, a drugi nije.