Grafovski algoritmi - čas 6

Svi najkraći putevi

Razmotrimo problem izračunavanja najkraćih puteva između *svaka dva* čvora u 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 zovemo dužinama. Ovaj problem nazivamo problem nalaženja *svih najkraćih puteva*. Za početak ćemo se zadovoljiti nalaženjem dužina svih najkraćih puteva, umesto samih puteva. Pretpostavimo da je graf usmeren; sve što će biti rečeno važi i za neusmerene grafove.

Kao i obično, pokušajmo sa direktnim induktivnim pristupom. Može se koristiti indukcija po broju grana ili čvorova.

Razmotrimo najpre indukciju po broju grana. Kako se menjaju najkraći putevi u grafu posle dodavanja nove grane (u, w) u graf? Nova grana može pre svega da predstavlja kraći put između čvorova u i w. Pored toga, može se promeniti 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, grane (u, w) i dužine najkraćeg puta od w do 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 je $O(|V|^4)$.

Razmotrimo sada indukciju prema broju čvorova. Kako se menjaju najkraći putevi ako se u graf doda novi čvor u? Potrebno je najpre pronaći dužine najkraćih puteva od u do svih ostalih čvorova, i od svih ostalih čvorova do 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, koja je već poznata. Potrebno je dakle da uporedimo ove zbirove za sve grane susedne sa u, i da među njima izaberemo najmanju. Najkraći put od w do u može se pronaći na sličan način. 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 dužina najkraćeg puta od v_1 do u i dužine najkraćeg puta od u do v_2. To je ukupno

$O(|V|^2)$ provera i sabiranja posle dodavanja svakog novog čvora, pa je složenost ovakvog algoritma u najgorem slučaju $O(|V| \cdot |V|^2) = O(|V|^3)$. Ispostavlja se da je u ovom slučaju indukcija po broju čvorova efikasnija nego indukcija po broju grana. Međutim, postoji još bolja induktivna konstrukcija za rešavanje ovog problema.

Ideja je da se ne menja broj čvorova ili grana, 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. Numerišimo čvorove na proizvoljan način brojevima od 1 do $|V|$. Put od u do w zove se *k-put* ako su redni brojevi svih čvorova na putu (izuzev u i w) manji ili jednaki od k. Specijalno, 0-put se sastoji samo od jedne grane (pošto se ni jedan drugi čvor ne može pojaviti na putu).

Induktivna hipoteza: Umemo da odredimo dužine najkraćih puteva između svaka dva čvora, pri čemu su dozvoljeni samo k-putevi, za $k < m$.

Baza indukcije je slučaj $m = 1$, kad se razmatraju samo direktne grane i rešenje je očigledno. 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 i da proverimo da li oni poboljšavaju k-puteve za $k < m$. Neka je v_m čvor sa rednim brojem m. Proizvoljan najkraći m-put sadrži v_m najviše jednom (pretpostavka je da ne postoji ciklus negativne dužine). Najkraći m-put između u i v može da se sastoji od najkraćeg $(m-1)$-puta od u do v_m, i najkraćeg $(m-1)$-puta od v_m do v. Prema induktivnoj hipotezi mi već znamo dužine najkraćih k-puteva za $k < m$, pa je dovoljno da saberemo ove dve dužine (i zbir uporedimo sa dužinom najkraćeg $(m-1)$-puta od u do v) da bismo pronašli dužinu najkraćeg m-puta od u do v. Ovaj algoritam poznat je pod nazivom Flojd-Varšalov algoritam. On je nešto brži od prethodnog algoritma (za konstantni faktor) i lakše ga je realizovati.

Same najkraće puteve možemo rekonstruisati tako što ćemo u matrici *put* na poziciji $put[i][j]$ postaviti maksimalnu vrednost m tako da čvor v_m pripada najkraćem putu od čvora i do čvora j. Ispisivanje puta 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.

```
vector<vector<pair<int,int>>> listaSuseda {{{1,5}, {2,3}}, {{3,7}, {4,5}},
            {{5,3}}, {}, {{6,3}, {7,2}}, {{8,1}}, {}, {}, {}};

// 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)
```

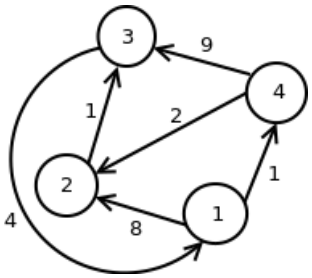
```
  if (put[i][j] == i)
    cout << i << " - ";
  else{
    odstampajPut(put, i, put[i][j]);
    odstampajPut(put, put[i][j], j);
  }
}

// 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] != numeric_limits<int>::max())
        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++)
      // za razlicite cvorove i,j ukoliko postoji put
      // od cvora i do cvora j stampamo ga
      if (i!=j && put[i][j] != -1){
         cout << "Najkraci put od cvora " << i
          << " do cvora " << j << " je: ";
         odstampajPut(put,i,j);
         // stampamo poslednji cvor na putu
         cout << j << endl;
      }
}

// funkcija koja racuna najkrace puteve izmedju svaka dva cvora
void sviNajkraciPutevi() {

  int brojCvorova = listaSuseda.size();
  vector<vector<int>> duzinaPuta(brojCvorova);
  vector<vector<int>> put(brojCvorova);

  // postavljamo sva rastojanja na beskonacno,
  // osim rastojanja cvora do samog sebe koje postavljamo na 0
  for (int i=0; i<brojCvorova; i++){
    duzinaPuta[i].resize(brojCvorova);
    put[i].resize(brojCvorova);
```

```
  for (int j=0; j<brojCvorova; j++){
    if (i==j){
       duzinaPuta[i][j] = 0;
       put[i][j] = 0;
    }
    else{
      duzinaPuta[i][j] = numeric_limits<int>::max();
      put[i][j] = -1;
    }
  }
}

// najkrace puteve izmedju dva cvora izmedju kojih
// postoji grana, postavljamo na tezinu grane,
// a cvor sa maksimalnim rednim brojem
// na putu izmedju ta dva cvora,
// s obzirom da je u pitanju direktna grana,
// postavljamo na pocetni cvor grane
for (int i=0; i<brojCvorova; i++)
  for (int j=0; j<listaSuseda[i].size(); j++){
    int k = listaSuseda[i][j].first;
    int tezina = listaSuseda[i][j].second;
    duzinaPuta[i][k] = tezina;
    put[i][k] = i;
  }

// proveravamo da li m-putevi skracuju puteve izmedju cvora i i j
for (int m=0; m<brojCvorova; m++)
  for (int i=0; i<brojCvorova; i++)
    for (int j=0; j<brojCvorova; j++)
        if (duzinaPuta[i][m]!=numeric_limits<int>::max() &&
            duzinaPuta[m][j]!=numeric_limits<int>::max() &&
            duzinaPuta[i][m]+duzinaPuta[m][j] < duzinaPuta[i][j]){
              // azuriramo duzinu najkraceg puta
              duzinaPuta[i][j] = duzinaPuta[i][m]+duzinaPuta[m][j];
              // postavljamo da se na putu od cvora i do cvora j
              // javlja i cvor sa oznakom m
              put[i][j] = m;
        }

// ukoliko je neki najkraci put od cvora do njega samog postavljen
// na negativnu vrednost, onda graf sadrzi ciklus negativne tezine
for (int i=0; i<brojCvorova; i++)
  if (duzinaPuta[i][i]<0){
    cout << "Graf sadrzi ciklus negativne tezine" << endl;
    return;
```

```
    }
  odstampajPuteve(duzinaPuta, put, brojCvorova);
}

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

Napomenimo da je redosled petlji važan, odnosno da je neophodno da spoljašnja petlja kontroliše parametar m koji ograničava tip dozvoljenih puteva. Unutrašnje dve petlje koriste se za proveru svih *parova* čvorova. Zapaža se da se ova provera može izvršavati sa parovima čvorova proizvoljnim redosledom, jer je svaka provera 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. Ukoliko polazni graf ima ciklus negativne težine, to se može utvrditi tako što će nakon izvršavanja Flojd-Varšalovog algoritma najkraći put od nekog čvora do njega samog biti manje od 0.

Složenost: Za svako m 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 algoritma za nalaženje dužina najkraćih puteva od jednog čvora $O((|E|+|V|)\log|V|)$. Ako je graf gust, pa je broj grana $\Omega(|V|^2)$, onda je opisani 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 $O(|V|(|E|+|V|)\log|V|)$ koja potiče od $|V|$ puta upotrebljenog algoritma za najkraće puteve od jednog čvora. Svakako jedna od prednosti Flojd-Varšalovog algoritma je i njegova jednostavna realizacija.



Slika 1: Težinski usmereni graf G.

Razmotrimo postupak određivanja svih najkraćih puteva za graf sa slike 1: on bi se sastojao iz narednih koraka:

$m = 0$:
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{pmatrix} \end{array}$$

$m = 1$:
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{pmatrix} \end{array}$$

$m = 2$:
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 8 & 9 & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 3 & 0 \end{pmatrix} \end{array}$$

$m = 3$:
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 8 & 9 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 12 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{pmatrix} \end{array}$$

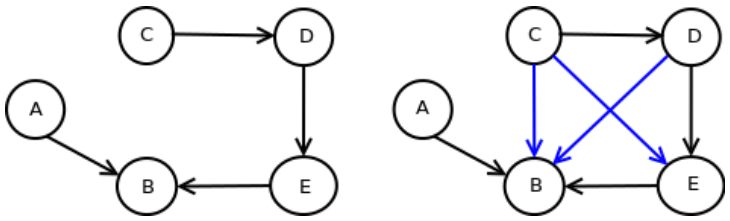
$m = 4$:
$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 3 & 4 & 1 \\ 5 & 0 & 1 & 6 \\ 4 & 7 & 0 & 5 \\ 7 & 2 & 3 & 0 \end{pmatrix} \end{array}$$

Tranzitivno zatvorenje

Za zadati usmereni graf $G = (V, E)$ njegovo *tranzitivno zatvorenje* $C = (V, F)$ 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 u do w. Na primer, na slici 2 prikazan je jedan usmeren graf i njegovo tranzitivno zatvorenje.

Postoji mnogo primena tranzitivnog zatvorenja, pa je važno imati efikasan algoritam za njegovo nalaženje.

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



Slika 2: Graf i njegovo tranzitivno zatvorenje: plavom bojom istaknute su grane koje su dodate u polazni graf.

Ovaj problem rešićemo redukcijom (svođenjem) na drugi problem. Drugim rečima pokazaćemo kako se može proizvoljni ulaz za problem tranzitivno zatvorenje svesti na ulaz za drugi problem, koji umemo da rešimo; nakon toga rešenje drugog problema transformišemo u rešenje problema tranzitivnog zatvorenja. Problem o kome je reč je nalaženje svih najkraćih puteva.

Neka je $G' = (V, E')$ kompletni usmereni graf (graf kod koga za svaki par čvorova postoje obe grane, u oba smera). Grani $e \in E'$ dodeljuje se dužina 0 ako je $e \in E$, odnosno 1 u protivnom. Sada za graf G' rešavamo problem nalaženja svih najkraćih puteva. Ako u G postoji put između v i w, onda je u G' njegova dužina 0. Šta više, put između v i w u G postoji ako i samo ako je dužina najkraćeg puta između v i w u G' jednaka 0. Drugim rečima, rešenje problema svih najkraćih puteva neposredno se transformiše u rešenje problema tranzitivnog zatvorenja.

Nije teško prepraviti algoritam za sve najkraće puteve, tako da direktno rešava problem tranzitivnog zatvorenja.

```
vector<vector<int>> listaSuseda {{1, 2}, {3, 4}, {5}, {},
                    {6, 7}, {8}, {}, {}, {}};

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

  int brojCvorova = listaSuseda.size();
  vector<vector<bool>> tranzitivnoZatvorenje(brojCvorova);

  // postavljamo sve vrednosti na false, osim elemenata na dijagonali
  for (int i=0; i<brojCvorova; i++){
    tranzitivnoZatvorenje[i].resize(brojCvorova);
    for (int j=0; j<brojCvorova; j++){
      if (i==j)
         tranzitivnoZatvorenje[i][j] = true;
      else
         tranzitivnoZatvorenje[i][j] = false;
    }
  }

  // sigurno postoji put izmedju dva cvora
  // ako izmedju njih postoji direktna grana
  for (int i=0; i<brojCvorova; i++)
    for (int j=0; j<listaSuseda[i].size(); j++){
      int k = listaSuseda[i][j];
      tranzitivnoZatvorenje[i][k] = true;
    }

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

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

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

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, `tranzitivnoZatvorenje[i,m]` i `tranzitivnoZatvorenje[m,j]`. Nešto 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 i i m, a druga zavisi samo od m i j. Zbog toga se prva provera može za fiksirane 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

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

```
vector<vector<int>> listaSuseda {{1, 2}, {3, 4}, {5}, {},
                    {6, 7}, {8}, {}, {}, {}};

// funkcija koja racuna najkrace puteve izmedju svaka dva cvora
void izracunajTranzitivnoZatvorenje() {

  int brojCvorova = listaSuseda.size();
  vector<vector<bool>> tranzitivnoZatvorenje(brojCvorova);

  // postavljamo sve vrednosti na false, osim elemenata na dijagonali
  for (int i=0; i<brojCvorova; i++){
    tranzitivnoZatvorenje[i].resize(brojCvorova);
```

```
    for (int j=0; j<brojCvorova; j++){
      if (i==j)
         tranzitivnoZatvorenje[i][j] = true;
      else
         tranzitivnoZatvorenje[i][j] = false;

    }
  }

  // postavljamo da postoji put izmedju dva cvora
  // ako izmedju njih postoji direktna grana
  for (int i=0; i<brojCvorova; i++)
    for (int j=0; j<listaSuseda[i].size(); j++){
      int k = listaSuseda[i][j];
      tranzitivnoZatvorenje[i][k] = true;
    }

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

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

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

Ovaj algoritam može se dalje usavršiti. Linija

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

može se ekvivalentno zameniti linijom

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

Zapaža se da se posle ove zamene u unutrašnjoj petlji algoritma operacija `or` primenjuje na vrstu i i vrstu m matrice $tranzitivnoZatvorenje$, a rezultat je nova vrsta i. Zbog toga, ako je $n = brojCvorova \leq 64$, vrste matrice $tranzitivnoZatvorenje$ mogu se predstaviti n-bitnim celim brojevima, pa se primena operacije `or` na vrste zamenjuje bitskom `or` operacijom dva cela broja, što je n puta brže. Ako je n veliko, 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.

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

Putevi i ciklusi u grafovima

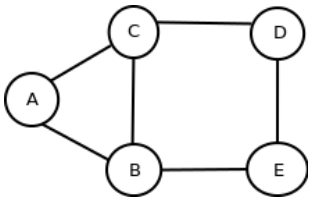
U ovom poglavlju razmotrićemo dve vrste puteva u grafovima:

- Ojlerov put, koji prolazi svakom granom grafa tačno jednom
- Hamiltonov put, koji posećuje svaki čvor u grafu tačno jednom

Iako možda na prvi pogled ove dve vrste puteva deluju slično, videćemo da su algoritmi koji se bave njima veoma različiti. Preciznije, pokazuje se da postoji jednostavna karakterizacija da li graf sadrži Ojlerov put i efikasan algoritam koji takav put pronalazi u grafu ako on postoji, dok je ispitivanje da li graf sadrži Hamiltonov put NP-težak problem i nije poznat efikasan način za njegovo rešavanje.

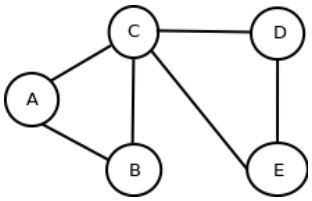
Ojlerovi putevi i ciklusi

Ojlerov put je put u grafu koji prolazi kroz svaku granu grafa tačno jednom. Na primer, u grafu prikazanom na slici 3 postoji Ojlerov put (C, D), (D, E), (E, B), (B, C), (C, A), (A, B).



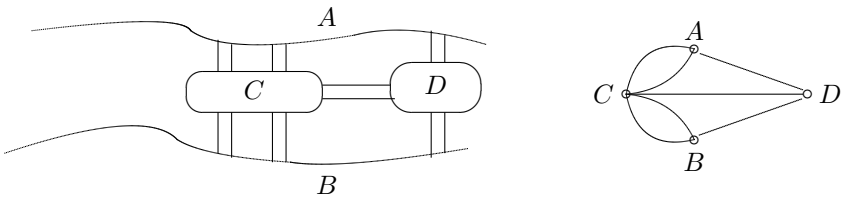
Slika 3: Graf koji sadrži Ojlerov put, ali ne sadrži Ojlerov ciklus.

Ojlerov ciklus je Ojlerov put čiji se početni i krajnji čvor poklapaju. U prethodnom grafu ne postoji Ojlerov ciklus, dok u grafu prikazanom na slici 4 postoji Ojlerov ciklus (C, D), (D, E), (E, C), (C, A), (A, B), (B, C).



Slika 4: 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 na sledeći zadatak 1736. godine. Grad Kenigsberg, danas Kalinjingrad, leži na obalama i na dva ostrva na reci Pregel, kao što je prikazano na slici 5. 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 ekvivalentno formulisati kao sledeći problem iz teorije grafova: da li je moguće u povezanom grafu pronaći ciklus, koji svaku granu sadrži tačno jednom – *Ojlerov ciklus*. Ili: da li je moguće nacrtati graf sa slike 5 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, 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. Takvi grafovi zovu se *Ojlerovi grafovi*. Pošto "graf" na slici 5 ima čvorove neparnog stepena, zaključujemo da problem Kenigsberških mostova nema rešenje.



Slika 5: Problem Kenigsberških mostova, i odgovarajući multigraf.

Dakle važi da neusmereni graf ima Ojlerov put samo 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

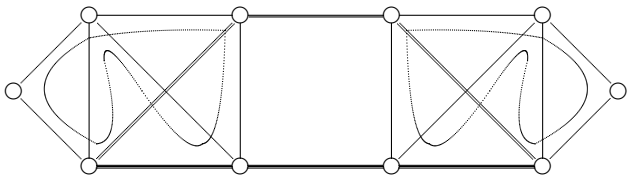
Ako važi prva stavka onda je svaki Ojlerov put istovremeno i Ojlerov ciklus. Ukoliko važi druga stavka, čvorovi neparnog stepena su početni i krajnji čvor Ojlerovog puta i u ovakvom grafu ne postoji Ojlerov ciklus. Na primer u grafu sa slike 3 jedino su čvorovi B i C neparnog stepena (dok su ostali čvorovi parnog stepena) te ova dva čvora predstavljaju početni i krajnji čvor Ojlerovog puta, a u njemu ne postoji Ojlerov ciklus. Dokažimo ovo tvrđenje.

Problem: U zadatom neusmerenom povezanom grafu $G = (V, E)$ čiji svi čvorovi imaju paran stepen, pronaći zatvoreni put (ciklus) P, takav da se u njemu svaka grana iz E pojavljuje tačno jednom.

Lako je pokazati da ako u grafu postoji Ojlerov ciklus, onda svi čvorovi grafa moraju imati paran stepen. 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, moramo najpre da izaberemo parametar po kome će biti izvedena indukcija. Taj izbor treba da omogući smanjivanje problema, bez njegove promene. Ako uklonimo čvor iz grafa, stepeni čvorova u dobijenom grafu nisu više svi parni. Trebalo bi da uklonimo takav skup grana S, da za svaki čvor v grafa broj grana iz 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 graf sa $< m$ grana čiji svi čvorovi imaju paran stepen, sadrži Ojlerov ciklus, koji se može pronaći.



Slika 6: 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.

Posmatrajmo graf $G = (V, E)$ sa m grana. Neka je P neki ciklus u G, i neka je G' graf dobijen uklanjanjem grana koje čine P iz grafa G. Stepeni svih čvorova u 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, videti primer na slici 6. 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 (njihov ukupan broj grana manji je od m). Prema tome, induktivna hipoteza može se primeniti na sve komponente: u svakoj komponenti G'_i postoji Ojlerov ciklus P'_i, i mi znamo da ga pronađemo. Potrebno je sada sve ove cikluse objediniti 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 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 polaznu tačku. U tom trenuku sve grane grafa G prođene su tačno jednom, što znači da je konstruisan Ojlerov ciklus. Ovaj dokaz daje efikasan algoritam za nalaženje Ojlerovog ciklusa u grafu.

U usmerenim grafovima važi da postoji Ojlerov put ako i samo ako je graf povezan i važi jedan od narednih uslova:

- ulazni i izlazni stepeni svakog čvora 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

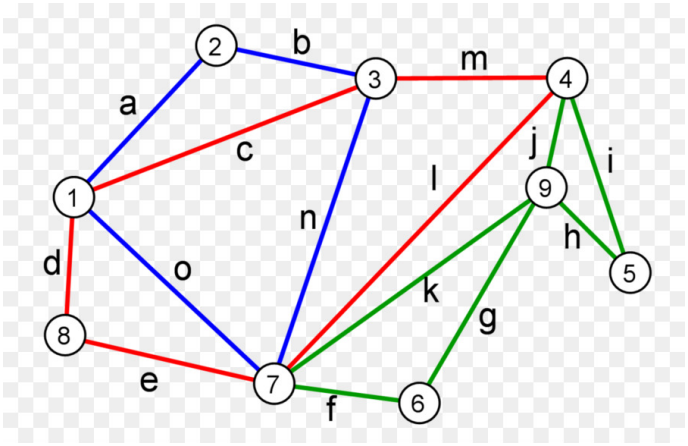
U prvom slučaju, svaki Ojlerov put je i Ojlerov ciklus, a u drugom slučaju Ojlerov put počinje u čvoru čiji je izlazni stepen veći a završava se u čvoru čiji je ulazni stepen veći.

Jedan efikasan način za konstruisanje Ojlerovog ciklusa u Ojlerovom grafu predstavlja Hirholcerov algoritam, zasnovan na prethodnom dokazu. Algoritam se sastoji iz nekoliko etapa, pri čemu se u svakoj etapi dodaju nove grane u ciklus. Najpre se konstruiše neki ciklus koji ne mora nužno da sadrži sve grane grafa, a zatim se on proširuje dodavanjem novih ciklusa u dati ciklus. Postupak se zaustavlja kada se sve grane dodaju u ciklus.

Ciklus se proširuje na sledeći način: pronalazi se čvor x koji pripada tekućem ciklusu ali koji ima izlaznu granu koja nije uključena u ciklus. Tom granom se kreće u otkrivanje novog puta koji se sastoji isključivo od čvorova koji nisu još u ciklusu. Put će se pre ili kasnije vratiti u čvor x i time će biti otkriven novi ciklus koji dodajemo u tekući ciklus.

Ako graf sadrži samo Ojlerov put, a ne i Ojlerov ciklus, može se iskoristiti isti algoritam tako što se u graf doda grana kojom se postiže uslov da graf ima Ojlerov ciklus, a onda se nakon pronalaska Ojlerovog ciklusa ta grana uklanja iz ciklusa.

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

void ispisiOjlerovCiklus(){

  stack<int> tekuciPut;
  vector<int> ojlerovCiklus;

  tekuciPut.push(0);
  int tekuciCvor = 0;

  while (!tekuciPut.empty()){
```



Slika 7: Ilustracija Hirholcerovog algoritma. Prvo na primer idemo granama a, b, n, o i vraćamo se u čvor 1, nakon toga vidimo da iz čvora 1 postoji neposećena izlazna grana ka čvoru 3 pa recimo obilazimo ciklus koji čine grane c, m, l, e, d i vraćamo se u čvor 1. Više nema neposećenih grana susednih čvoru 1 i vraćamo se unazad i razmatramo redom čvorove 8, 7 i vidimo da iz čvora 7 postoji ciklus sačinjen od grana k, j, i, h, g, f. Nakon toga vraćamo se unazad skupom čvorova i dobijamo Ojlerov ciklus: $1, 8, 7, 6, 9, 5, 4, 9, 7, 4, 3, 1, 7, 3, 2, 1$.

```
// ako iz tekuceg cvora postoji jos neka grana
if (listaSuseda[tekuciCvor].size()){

  // tekuci cvor dodajemo u put
  tekuciPut.push(tekuciCvor);

  // pronalazimo cvor do koga postoji grana iz tekuceg cvora
  // uzimamo poslednji 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;
}
// vracamo se unazad da bismo nasli preostale cikluse
else{

  ojlerovCiklus.push_back(tekuciCvor);
  tekuciCvor = tekuciPut.top();
  tekuciPut.pop();
```

```
    }
  }

  // stampamo pronadjeni Ojlerov ciklus u suprotnom redosledu
  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;
}
```

Vreme izvršavanja ovog algoritma je $O(|E|)$.

Hamiltonovi ciklusi

Hamiltonov put je put koji posećuje svaki čvor grafa tačno jednom. Na primer, graf prikazan na slici 3 sadrži Hamiltonov put (A, B), (B, C), (C, D), (D, E).

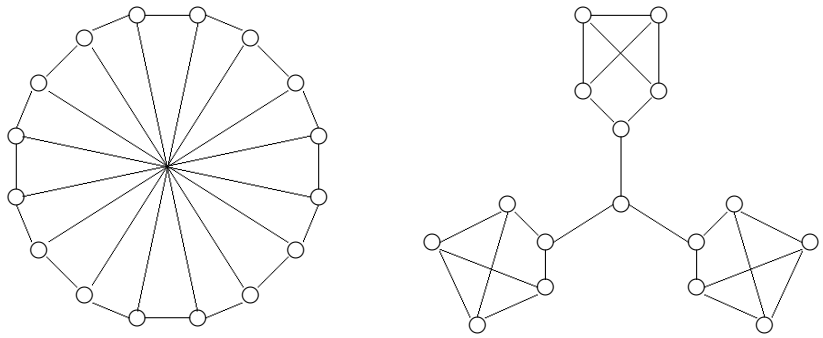
Ako Hamiltonov put počinje i završava se u istom čvoru on se naziva *Hamiltonov ciklus*. Prethodno pomenuti graf ima i Hamiltonov ciklus koji počinje i završava se u čvoru A: (A, B), (B, E), (E, D), (D, C), (C, A). Graf koji sadrži Hamiltonov ciklus zove se *Hamiltonov graf*.

Ova vrsta ciklusa nazvana je tako u cast Vilijama Hamiltona koji je 1857. godine izumeo jednu vrstu slagalice koja uključuje potragu za ovom vrstom ciklusa u grafu sačinjenom od ivica dodekaedra.

Problem ima usmerenu i neusmerenu verziju; mi ćemo se baviti samo neusmerenom varijantom.

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 8 imaju po 16 čvorova stepena 3 (dakle imaju iste stepene čvorova), ali je prvi Hamiltonov, a drugi očigledno nije. Ovaj problem spada u klasu NP-kompletnih problema.

Primetimo da ako je graf kompletan, onda on sigurno sadrži Hamiltonov ciklus. Jasno je i to da što je graf "gušći" (što je veći broj grana) postoji veća verovatnoća



Slika 8: Dva grafa sa po 16 čvorova stepena 3, od kojih je prvi Hamiltonov, a drugi nije.

pronalaženja Hamiltonovog ciklusa. S tim u vezi dokazane su naredne teoreme:

Oreova teorema: Neka je $G = (V, E)$ neusmeren graf za koji važi $|V| = n \geq 3$. Ako je zbir stepena svaka dva nesusedna čvora u grafu bar n, graf sadrži Hamiltonov ciklus.

Posledica ove teoreme je naredna teorema.

Dirakova teorema: Ako je stepen svakog čvora u neusmerenom grafu bar $n/2$, graf sadrži Hamiltonov ciklus.

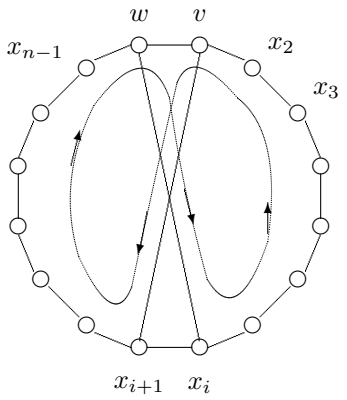
Neka je $G = (V, E)$ povezan neusmeren graf i neka za proizvoljan čvor $v \in V$ $d(v)$ označava njegov stepen. Dokažimo Oreovu teoremu.

Dokaz se zasniva na indukciji po broju grana koje treba ukloniti iz kompletnog grafa da bi se dobio zadati graf. Baza indukcije je kompletan graf. Svaki kompletan graf sa bar tri čvora sadrži Hamiltonov ciklus, koji je lako pronaći.

Induktivna hipoteza: Umemo da pronađemo Hamiltonov ciklus u grafovima koji zadovoljavaju navedene uslove ako imaju bar $n(n-1)/2 - m$ grana.

Sada treba da pokažemo kako pronaći Hamiltonov ciklus u grafu sa $n(n-1)/2 - (m+1)$ grana koji zadovoljava uslove problema. Neka je $G = (V, E)$ takav graf. Izaberimo proizvoljna dva nesusedna čvora v i w u G (to je moguće ako graf nije kompletan), i posmatrajmo graf G' koji se od G dobija dodavanjem grane (v, w). Prema induktivnoj hipotezi mi umemo da pronađemo Hamiltonov ciklus u grafu G'. Označimo sa $x_1, x_2, \ldots, x_n$ sve čvorove u grafu G' i neka je x_1, x_2, $\ldots$, x_n, x_1 takav ciklus u G' (videti sliku 9). Ako grana (v, w) nije deo ciklusa, onda je isti ciklus deo grafa G, pa je problem rešen. U protivnom, bez smanjenja opštosti može se pretpostaviti da je $v = x_1$ i $w = x_n$. Prema datim uslovima je $d(v) + d(w) \geq n$.

Neka je



Slika 9: Modifikacija Hamiltonovog ciklusa posle izbacivanja grane (v, w).

$A = \{i : 1 \leq i \leq n-1 | x_{i+1}$ susedno sa $v\}$,

$B = \{i : 1 \leq i \leq n-1 | x_i$ susedno sa $w\}$.

Važi da je $|A| = d(v)$ i $|B| = d(w)$, odnosno da je $|A| + |B| \geq n$. A i B su podskupovi skupa $\{1, 2, \ldots, n-1\}$ koji ima $n-1$ elemenata te ne mogu biti disjunktni, te sigurno postoji neko $i \in A \cap B$, odnosno u grafu G za neko i postoje grane (w, x_i) i (v, x_{i+1}). Od ovih dveju grana može se formirati novi Hamiltonov ciklus $v(= x_1), x_{i+1}, x_{i+2}, \ldots, w(= x_n), x_i, x_{i-1}, \ldots, v$, koji ne sadrži granu (v, w), videti sliku 9.

Direktna primena ovog dokaza prilikom implementacije algoritma polazi od kompletnog grafa, iz koga se jedna za drugom izbacuju grane koje ne pripadaju zadatom grafu. Bolje je rešenje početi sa mnogo manjim grafom, na sledeći način. U datom grafu G najpre pronalazimo dugačak put (na primer, pomoću DFS), a onda dodajemo nove grane tako da put produžimo do Hamiltonovog ciklusa. Tako je dobijen veći graf G'. Obično je dovoljno dodati samo nekoliko grana. Međutim, čak i u najgorem slučaju biće dodata najviše $n-1$ grana. Polazeći od G', dokaz teoreme se zatim primenjuje iterativno, sve dok se ne pronađe Hamiltonov ciklus u G. Ukupan broj koraka za zamenu jedne grane je $O(n)$. Potrebno je zameniti najviše $n-1$ grana, pa je vremenska složenost algoritma $O(n^2)$.