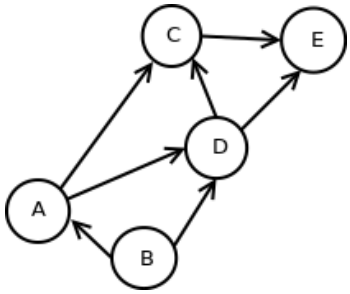
Grafovski algoritmi - čas 4

Topološko sortiranje

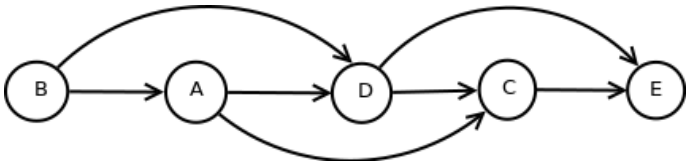
Pretpostavimo da je zadat skup poslova u vezi sa čijim redosledom izvršavanja postoje neka ograničenja. Neki poslovi zavise od drugih, odnosno ne mogu se započeti pre nego što se ti drugi poslovi završe. Sve zavisnosti su poznate, a cilj je napraviti takav redosled izvršavanja poslova koji zadovoljava sva zadata ograničenja; drugim rečima, traži se takav redosled izvršavanja za koji važi da svaki posao započinje tek kad su završeni svi poslovi od kojih on zavisi. Ovaj problem je važan u raznim domenima, kao što je određivanje redosleda u kom je potrebno izvršiti ponovno izračunavanje vrednosti formula u programima za tabelarna izračunavanja, redosled u kom treba izvršiti zadatke u mejkfajlu, unapređenje paralelizma instrukcija i slično. Zadatak koji želimo da rešimo jeste konstruisati efikasan algoritam za formiranje takvog redosleda. Ovaj problem može se formulisati kao grafovski i naziva se *topološko sortiranje grafa*. Zadatim poslovima i njihovim međuzavisnostima može se na prirodan način pridružiti usmereni graf. Svakom poslu pridružuje se čvor, a usmerena grana od posla x do posla y postoji ako se posao y ne može započeti pre završetka posla x. Jasno je da graf mora biti bez usmerenih ciklusa, jer se u protivnom neki poslovi nikada ne bi mogli započeti.



Slika 1: Usmereni aciklički graf u kojem postoji tačno jedno topološko uređenje.

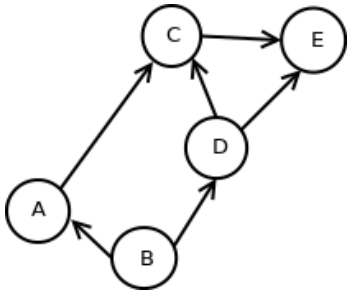
Problem: U zadatom usmerenom acikličkom grafu $G = (V, E)$ sa n čvorova numerisati čvorove brojevima od 1 do n, tako da ako je proizvoljan čvor v numerisan brojem k, onda su svi čvorovi do kojih postoji usmerena grana iz čvora v numerisani brojevima većim od k.

Na primer, u grafu prikazanom na slici 1 postoji samo jedno ispravno topološko uređenje čvorova i to je B, A, D, C, E. Graf sa slike 1 možemo predstaviti i drugačije, tako da čvorovi budu poređani u topološkom poretku, a onda su grane uvek usmerene od jednog čvora ka čvoru koji je desno od njega.



Slika 2: Usmereni aciklički graf kod koga su čvorovi poređani u redosledu topološkog uređenja.

U opštem slučaju, može postojati veći broj ispravnih topoloških uređenja čvorova. Ako razmotrimo graf sa slike 3, u njemu postoje dva ispravna topološka uređenja: B, A, D, C, E i B, D, A, C, E.



Slika 3: Usmereni aciklički graf u kojem postoje dva različita topološka uređenja.

Razmotrićemo dva različita algoritma za rešavanje ovog problema.

Kanov algoritam

Prirodna je sledeća induktivna hipoteza: umemo da numerišemo na zahtevani način čvorove svih usmerenih acikličkih grafova sa manje od n čvorova.

Bazni slučaj je slučaj grafa koji sadrži tačno jedan čvor, odnosno posao, i on se trivijalno rešava. Kao i obično, posmatrajmo proizvoljni graf sa n čvorova, uklonimo jedan čvor iz njega, primenimo induktivnu hipotezu na preostale čvorove u grafu i pokušajmo da proširimo numeraciju na polazni graf. Ono što je važno primetiti jeste da imamo slobodu izbora čvora koji uklanjamo. Trebalo bi ga izabrati tako da što jednostavnije proširimo induktivnu hipotezu. Postavlja se pitanje koji čvor je najlakše numerisati? To je očigledno čvor (posao) koji ne zavisi od drugih poslova, odnosno čvor sa ulaznim stepenom nula; njemu se može dodeliti broj 1. Da li u proizvoljnom usmerenom acikličkom grafu uvek postoji čvor sa ulaznim stepenom nula? Intuitivno se nameće potvrdan odgovor, jer se sa označavanjem negde mora započeti. Sledeća lema potvrđuje ovu činjenicu.

Lema: Usmereni aciklički graf uvek ima čvor sa ulaznim stepenom nula.

Dokaz: Ako bi svi čvorovi grafa imali pozitivne ulazne stepene, mogli bismo da krenemo iz nekog čvora "unazad" prolazeći grane u suprotnom smeru. Međutim, broj čvorova u grafu je konačan, pa se u tom obilasku mora u nekom trenutku naići na neki čvor po drugi put, što znači da u grafu postoji usmereni ciklus. Ovo je suprotno pretpostavci da se radi o acikličkom grafu. Dakle u usmerenom acikličkom grafu uvek postoji čvor sa ulaznim stepenom nula.[1] □

Pretpostavimo da smo pronašli čvor sa ulaznim stepenom nula. Numerišimo ga sa 1, uklonimo sve grane koje vode iz njega, i numerišimo ostatak grafa (koji je takođe aciklički) brojevima od 2 do n (prema induktivnoj hipotezi oni se mogu numerisati brojevima od 1 do $n-1$, a zatim se svaki redni broj može povećati za jedan). Vidi se da je posle izbora čvora sa ulaznim stepenom nula, preostali problem sličan polaznom problemu.

Dakle, problem se može rešiti stalnim pronalaženjem čvorova sa ulaznim stepenom 0. Jedini problem pri realizaciji ovog algoritma je kako pronaći čvor sa ulaznim stepenom nula i kako popraviti ulazne stepene čvorova posle uklanjanja grana koje polaze iz datog čvora. Možemo alocirati niz `ulazniStepen` dimenzije jednake broju čvorova u grafu i inicijalizovati ga na vrednosti ulaznih stepena čvorova. Ulazne stepene čvorova u grafu možemo jednostavno odrediti prolaskom kroz skup svih grana proizvoljnim redosledom (sve grane su navedene u listi povezanosti kojom je predstavljen graf) i povećavanjem za jedan vrednosti `ulazniStepen[w]` svaki put kad se naiđe na neku granu (v, w). Čvorovi sa ulaznim stepenom nula stavljaju se u red (ili stek, što bi bilo jednako dobro). Prema prethodnoj lemi u grafu postoji bar jedan čvor sa ulaznim stepenom nula. Neka je v jedan od takvih čvorova. Čvor v se kao prvi u redu lako pronalazi; on se uklanja iz reda i numeriše brojem 1. Zatim se za svaku granu (v, w) koja izlazi iz čvora v vrednost `ulazniStepen[w]` smanjuje za jedan. Ako neka od tih vrednosti pri tome postane nula, čvor w upisuje se u red. Posle uklanjanja čvora v graf ostaje aciklički, pa u njemu prema prethodnoj lemi ponovo postoji čvor sa ulaznim stepenom nula. Algoritam završava sa radom kad red koji sadrži čvorove stepena nula postane prazan, jer su u tom trenutku svi čvorovi numerisani. Opisani algoritam zove se *Kanov algoritam*.

U tabeli 1 ilustrovano je izvršavanje Kanovog algoritma na primeru grafa sa slike 3 zadatog listom povezanosti kod koje su za svaki čvor susedi poređani leksikografski. Za svaki od koraka algoritma prikazane su trenutne vrednosti ulaznih stepena čvorova grafa, sadržaj reda koji sadrži čvorove ulaznog stepena nula koji još uvek nisu numerisani i poslednji numerisani čvor u grafu. U drugom koraku se moglo desiti da se u red doda čvor D, a zatim čvor A i u tom slučaju bilo bi dobijeno uređenje: B, D, A, C, E.

Ako bi nakon završetka rada algoritma za neke čvorove važilo da nisu bili dodati u red, to bi značilo da postoji podskup skupa čvorova takav da u odgovarajućem indukovanom podgrafu svi čvorovi imaju ulazni stepen veći od nula, što znači da

[1] Analogno bi se pokazalo da u usmerenom acikličkom grafu uvek postoji čvor sa izlaznim stepenom nula.

$d(A)$	$d(B)$	$d(C)$	$d(D)$	$d(E)$	Red	Naredni numerisani čvor
1	0	2	1	2	B	
0		2	0	2	A, D	$B : 1$
		1		2	D	$A : 2$
		0		1	C	$D : 3$
				0	E	$C : 4$
						$E : 5$

Table 1: Primer izvršavanja Kanovog algoritma za graf sa slike 3.

bi indukovani podgraf (a time i polazni graf) sadržao usmereni ciklus, suprotno pretpostavci da je graf aciklički.

Važno je napomenuti da u algoritmu ne moramo iz samog grafa izbacivati grane (odnosno menjati listu povezanosti kojom je graf zadat), već je jedino važno da za svaki čvor ažuriramo njegov ulazni stepen.

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

void topolosko_sortiranje(){

  int brojCvorova = listaSuseda.size();
  // niz koji cuva ulazne stepene cvorova
  vector<int> ulazniStepen(brojCvorova,0);
  // niz koji cuva redne brojeve cvorova u topoloskom uredjenju
  vector<int> topoloskoUredjenje;
  // broj posecenih cvorova
  int brojPosecenih = 0;

  // inicijalizujemo niz ulaznih stepena cvorova
  for (int i = 0; i < listaSuseda.size(); i++)
    for (int j = 0; j < listaSuseda[i].size(); j++)
      ulazniStepen[listaSuseda[i][j]]++;

  // red koji cuva cvorove ulaznog stepena nula
  queue<int> cvoroviStepenaNula;

  // cvorove koji su ulaznog stepena 0 dodajemo u red
  for (int i = 0; i < brojCvorova; i++)
    if (ulazniStepen[i] == 0)
       cvoroviStepenaNula.push(i);

  while(!cvoroviStepenaNula.empty()){
    // cvor sa pocetka reda numerisemo narednim brojem
    int cvor = cvoroviStepenaNula.front();
```

```
    cvoroviStepenaNula.pop();
    topoloskoUredjenje.push_back(cvor);

    brojPosecenih++;

    // za sve susede tog cvora azuriramo ulazne stepene
    for(int i = 0; i < listaSuseda[cvor].size(); i++){
      int sused = listaSuseda[cvor][i];
      ulazniStepen[sused]--;
      // ako je ulazni stepen suseda postao 0, dodajemo ga u red
      if (ulazniStepen[sused] == 0)
         cvoroviStepenaNula.push(sused);
    }
  }

  if (brojPosecenih == brojCvorova){
     cout << "Redosled cvorova u topoloskom uredjenju je:" << endl;
     for(int i = 0; i < brojCvorova; i++)
        cout << topoloskoUredjenje[i] << ": " << i+1 << endl;
  }
  else
     cout << "Graf nije aciklicki" << endl;
}

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

Vremenska složenost izračunavanja početnih vrednosti elementa niza `ulazniStepen` je $O(|V| + |E|)$. U petlji `while` (kroz koju se prolazi $|V|$ puta) za nalaženje čvora sa ulaznim stepenom nula potrebno je konstantno vreme (pristup redu). Svaka grana (v, w) razmatra se tačno jednom, u petlji kroz koju se prolazi posle uklanjanja čvora v iz reda. Prema tome, ukupan broj promena vrednosti elemenata niza `ulazniStepen` jednak je broju grana u grafu. Vremenska složenost algoritma je dakle $O(|V| + |E|)$, odnosno linearna je funkcija od veličine grafa.

Algoritam zasnovan na DFS pretrazi

Kao što smo ranije zaključili u grafu važi:

- ako je grana (u, v) grana DFS drveta, direktna ili poprečna grana, za nju važi $u.Post > v.Post$,
- ako je grana (u, v) povratna grana, za nju važi $u.Post \leq v.Post$.

U usmerenom acikličkom grafu ne postoji ciklus, pa ne postoje povratne grane u odnosu na DFS drvo. Dakle, za svaku granu (u, v) grafa važi uslov $u.Post > v.Post$. Stoga, ako uredimo čvorove grafa u opadajućem redosledu u odnosu na odlaznu numeraciju čvorova, dobićemo topološko uređenje grafa. Ovo tvrđenje važi zato što će na ovaj način za proizvoljnu granu (u, v) acikličkog grafa važiti da je polazni čvor u numerisan manjom vrednošću od dolaznog čvora v, što je u skladu sa zahtevima problema koji rešavamo.

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

void dfs(int cvor, vector<bool> &posecen, vector<int> &odlazna){
  posecen[cvor] = true;

  // rekurzivno prolazimo kroz sve susede koje nismo obisli
  for (auto sused : listaSuseda[cvor]){
    if (!posecen[sused])
       dfs(sused,posecen,odlazna);
  }

  // u vektor odlazna dodajemo na kraj naredni cvor
  // koji napustamo pri DFS obilasku
  odlazna.push_back(cvor);
}

void topolosko_sortiranje(){

  int brojCvorova = listaSuseda.size();
  vector<bool> posecen(brojCvorova);
  // niz koji sadrzi redom cvorove prema redosledu napustanja
  vector<int> odlazna;

  for (int cvor=0; cvor<brojCvorova; cvor++)
    if (!posecen[cvor])
       dfs(cvor,posecen,odlazna);

  // cvorove ispisujemo u opadajućem redosledu odlazne numeracije
  cout << "Redosled cvorova u topoloskom uredjenju je:" << endl;
  for(int i = brojCvorova-1; i >= 0; i--)
    cout << odlazna[i] << ": " << brojCvorova-i << endl;

}

int main(){
  topolosko_sortiranje();
  return 0;
```

```
}
```

S obzirom na to da se ova implementacija svodi na DFS pretragu i određivanje odlazne numeracije čvorova, vremenska složenost ovog algoritma je $O(|E| + |V|)$.