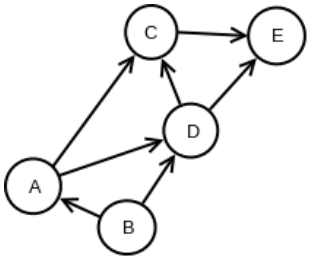
Topološko sortiranje

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 redosled izvršavanja za koji važi da svaki posao započinje tek kad su završeni svi poslovi od kojih on zavisi. Na primer, želimo da sagradimo kuću. Da bi to uspeli potrebno je da iskopamo temelj, da sazidamo zidove, da stavimo krov, da uvedemo struju i vodu. Pritom, naravno, nije moguće npr. sazidati zidove dok se ne stavi temelj, niti uvesti vodu dok se ne stavi krov na kuću. Potrebno je odrediti neki ispravan redosled izvršavanja ovih poslova.

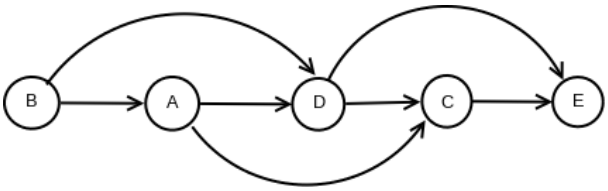
Dati problem ima primenu u raznim domenima: na primer, za određivanje redosleda u kom je potrebno izvršiti ponovno izračunavanje vrednosti formula u programima za tabelarna izračunavanja, utvrđivanje redosleda u kom treba izvršiti zadatke u mejkfajlu, unapređenje paralelizma instrukcija i slično. Želimo da osmislimo efikasan algoritam za formiranje takvog redosleda. Ovaj problem može se formulisati kao grafovski i naziva se *topološko sortiranje grafa* (eng. topological sort). Naime, zadatim poslovima i njihovim međuzavisnostima može se na prirodan način pridružiti usmereni graf: svakom poslu pridružuje se čvor, a usmerena grana od čvora x do čvora y postoji ako se posao y ne može započeti pre završetka posla x. Zadatak je odrediti poredak čvorova tako da za svaku granu grafa važi da je polazni čvor grane numerisan manjom vrednošću nego završni čvor te grane. Graf nad kojim razmatramo ovaj problem 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 čvorova.

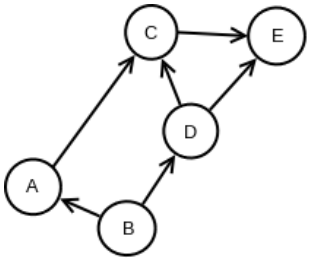
Problem: U zadatom usmerenom acikličkom grafu $G = (V, E)$ sa n čvorova numerisati čvorove brojevima od 1 do n, tako da ako je proizvoljan čvor v numerisan brojem k, onda su svi čvorovi do kojih postoji usmerena grana iz čvora v numerisani brojevima većim od k.

Na primer, u grafu prikazanom na slici 1 postoji samo jedno ispravno topološko uređenje čvorova i to je B, A, D, C, E. Čvor D, recimo, mora da bude numerisan većim brojem od čvorova B i A jer postoje grane do D iz čvorova A i B. Slično čvor D mora biti numerisan manjim brojem od čvorova C i E jer postoje grane od čvora D do čvorova C i E. Dakle, u ovom grafu redni broj čvora D mora biti 3. Graf sa slike 1 možemo predstaviti i pogodnije, tako da čvorovi budu poređani duž jedne prave uređeni u odnosu na topološki poredak. Onda su grane grafa uvek usmerene udesno (slika 2).



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

U opštem slučaju može postojati veći broj ispravnih topoloških uređenja grafa. Ako razmotrimo graf sa slike 3, u njemu postoje dva ispravna topološka uređenja: B, A, D, C, E i B, D, A, C, E. Oni su prikazani na slici 4.



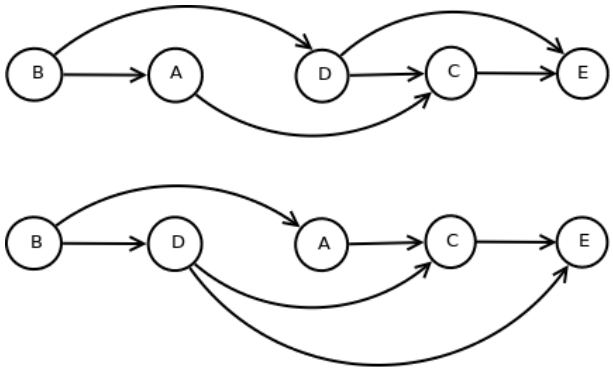
Slika 3: Usmereni aciklički graf u kojem postoje dva različita topološka uređenja čvorova.

Razmotrićemo dva različita algoritma za određivanje topološkog uređenja u acikličkom usmerenom grafu: Kanov algoritam i algoritam zasnovan na pretrazi grafa u dubinu.

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 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



Slika 4: Dva moguća topološka uređenja grafa sa slike 3.

da proširimo numeraciju na polazni graf. Ono što je važno primetiti jeste da imamo slobodu izbora čvora koji uklanjamo. Treba izabrati čvor 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. Postavlja se pitanje 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 ulaznog stepena 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 čiji je ulazni stepen nula.[1] □

Pretpostavimo da smo pronašli čvor čiji je ulazni stepen 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. Primetimo da je posle izbora čvora sa ulaznim stepenom nula, preostali problem sličan polaznom problemu. Napomenimo još u ovom trenutku da nije neophodno efektivno izbacivati grane iz grafa, jer je to operacija koja se ne izvršava efikasno ako je graf predstavljen listom povezanosti, već da je isti efekat moguće izvesti efikasnije.

Dakle, problem se može rešiti sukcesivnim pronalaženjem čvorova sa ulaznim stepenom 0. Jedini problem pri realizaciji ovog algoritma je kako efikasno pronaći

[1] Analogno bi se pokazalo da u usmerenom acikličkom grafu uvek postoji čvor izlaznog stepena nula.

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 u proizvoljnom redosledu i povećavanjem za jedan vrednosti `ulazniStepen[w]` svaki put kad se naiđe na neku granu (v, w). Ukoliko je graf zadat listom povezanosti, sve grane su navedene u listi povezanosti kojom je predstavljen i ovaj korak biće linearne složenosti po broju grana u grafu. Tokom izvršavanja algoritma bitno je održavati spisak čvorova čiji je ulazni stepen 0 jer ih je potrebno obraditi u nekom poretku – primetimo da takvih čvorova u svakom od koraka može biti puno. Dakle, potrebno je čvorove sa ulaznim stepenom nula čuvati u nekoj kolekciji u koju je efikasno umetati i iz koje je efikasno uklanjati elemente. U ove svrhe može se koristiti red (ili stek, što bi bilo jednako dobro). Prema prethodnoj lemi u polaznom acikličkom 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 polazi iz čvora v vrednost `ulazniStepen[w]` smanjuje za jedan. Time se evidentira da zavisnosti koje potiču od trenutno numerisanog čvora više nisu od značaja: svakako će svi preostali čvorovi biti numerisani većim vrednostima od tekuće. Ako vrednost ulaznog stepena čvora w 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 kada red koji sadrži čvorove stepena nula postane prazan, jer su u tom trenutku svi čvorovi numerisani. Opisani algoritam zove se *Kanov algoritam*, po njegovom autoru Arturu Kanu.

U tabeli 1 ilustrovano je izvršavanje Kanovog algoritma na primeru grafa sa slike 3. Pretpostavićemo da je graf zadat listama povezanosti, tako da su za svaki čvor njegovi susedi poređani u leksikografski rastućem poretku. 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. Primetimo da se u drugom koraku moglo desiti da se u red najpre doda čvor D, a zatim čvor A i u tom slučaju bilo bi dobijeno drugačije topološko uređenje: B, D, A, C, E.

$d(A)$	$d(B)$	$d(C)$	$d(D)$	$d(E)$	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.

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 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 vrednost njegovog ulaznog stepena.

```
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);
    }
  }

  // ako smo numerisali sve cvorove u grafu
  if (brojPosecenih == brojCvorova){
     // stampamo dobijeno topolosko uredjenje
     cout << "Redosled cvorova u topoloskom uredjenju je:" << endl;
     for(int i = 0; i < brojCvorova; i++)
        cout << topoloskoUredjenje[i] << ": " << i+1 << endl;
  }
  else
     // zakljucujemo da graf sadrzi usmereni ciklus
     cout << "Graf nije aciklicki" << endl;
}

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

Vremenska složenost inicijalizacije niza `ulazniStepen` je $O(|V|+|E|)$, u slučaju kada je graf zadat listama povezanosti. U petlji `while` (kroz koju se prolazi $|V|$ puta) za pronalaženje 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 nakon uklanjanja čvora v iz reda. Prema tome, ukupan broj promena vrednosti elemenata niza `ulazniStepen` u svim izvršavanjima spoljašnje `while` petlje jednak je broju grana u grafu. Vremenska složenost Kanovog algoritma je dakle $O(|V|+|E|)$, odnosno linearna je funkcija od veličine grafa.

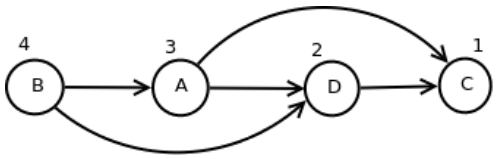
Algoritam zasnovan na DFS pretrazi

Kao što smo ranije zaključili u grafu $G = (V, E)$ važi:

- ako je grana $(u, v) \in E$ grana DFS drveta, direktna ili poprečna grana, za nju važi $u.Post > v.Post$,
- ako je grana $(u, v) \in E$ povratna grana u odnosu na DFS drvo, 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 $u.Post > v.Post$. Ako sa $n(x)$ označimo redni broj čvora x u topološkom poretku grafa G, za svaku granu (u, v) potrebno je da važi $n(u) < n(v)$. Odavde sledi da ako

čvorove grafa uredimo u opadajućem redosledu u odnosu na odlaznu numeraciju čvorova, dobićemo jedno 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.



Slika 5: Usmereni aciklički graf: uz svaki čvor prikazan je njegov broj pri odlaznoj DFS numeraciji.

Razmotrimo graf sa slike 5: on je usmeren i aciklički. Ako pokrenemo DFS pretragu iz čvora B redosled čvorova u kojima ćemo napuštati čvorove je C, D, A, B. Dakle, topološko uređenje grafa dobićemo obrtanjem ovog redosleda, odnosno redosled čvorova u topološkom poretku biće B, A, D, C. Primetimo da to odgovara i redosledu čvorova sleva nadesno u prikazu grafa kod koga su sve grane usmerene sleva udesno.

```
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 prikazani algoritam svodi na DFS pretragu i određivanje odlazne numeracije čvorova, njegova vremenska složenost iznosi $O(|E| + |V|)$.