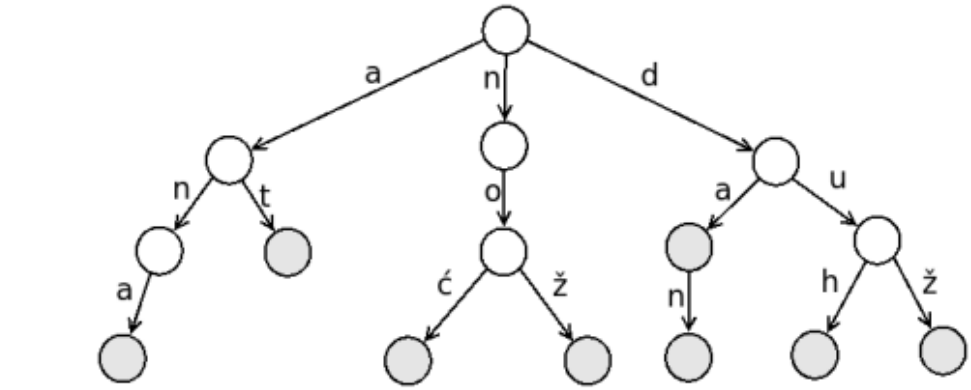
Konstrukcija i analiza algoritama

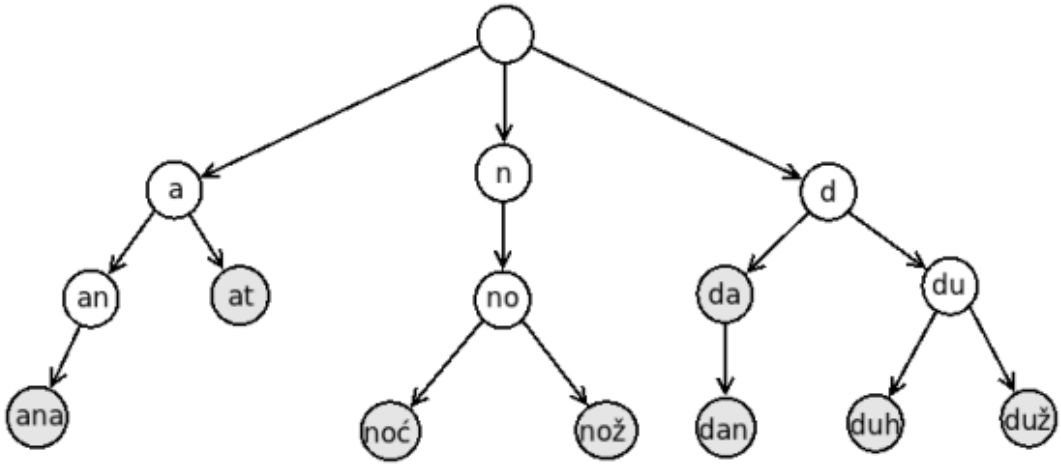
Minimalni nivo

1. Prefiksno drvo - pojam, svojstva, primene, primeri

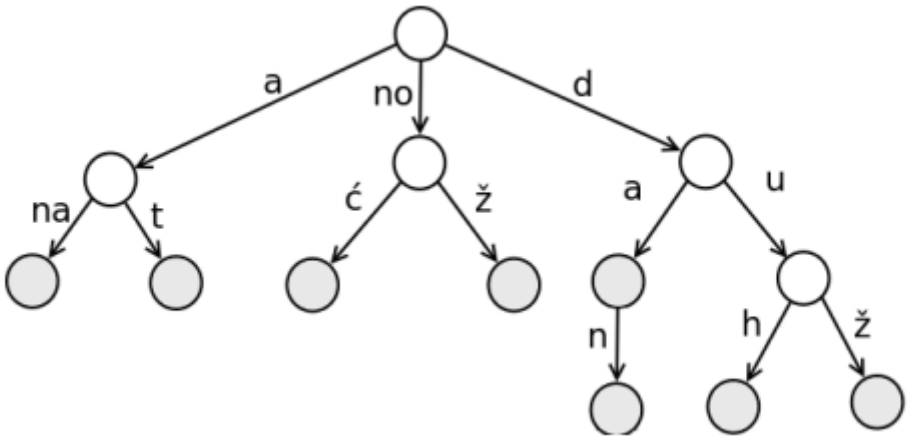
Pored uređenih binarnih drveta i heš tabela, još jedna drvolika struktura podataka pruža efikasan asocijativni pristup kada su ključevi niske. Ta struktura se naziva **prefiksno drvo** poznato pod nazivom **trie** (od reTRIEval). Osnovna ideja ove strukture je da se ključ pridružen svakom čvoru dobija nadovezivanjem karaktera koji se nalaze na granama duž putanje od korena do tog čvora. Koren sadrži praznu reč, a prelaskom preko svake grane se na do tada formiranu reč zdesna nadovezuje još jedan karakter.

Primer: reči *ana, at, noć, nož, da, dan, duh, duž* čuvamo u sledećem prefiksnom drvetu (slika 1), a čvorovi na kraju sadrže sledeće reči (slika 2):





Primećujemo da se većina ključeva nalazi u listovima, ali ne svi (ključ *da*). Zbog toga, u svakom čvoru čuvamo dodatnu informaciju da li je on ključ ili ne. Mana prefiksnih drveta je to što zauzimaju puno memorije, međutim moguće ih je malo optimizovati. U slučaju da neki čvor ima samo jednog potomka i ne predstavlja ključ onda se grana do njega i od njega mogu spojiti, a čvor ukloniti.



Primena prefiksnih drveta se pre svega ogleda u implementaciji skupova i mapa zbog asocijativnog pristupa podacima. Takođe, nalazi primene i u implementaciji konačnih rečnika (na primer automatsko kompletiranje ili provera ispravnosti reči).

2. Upiti raspona - pojam, osnovne tehnike rešavanja problema

Upiti raspona (range queries) su upiti koji zahtevaju izračunavanje statistika nekih raspona tj. segmenata niza.

Problem: Efikasno izračunati zbirove segmenata [a, b] nekog niza.

- Rešenje grubom silom koje bi za svaki upit računalo zbir od početka do kraja segmenta bi jasno bilo složenosti O(mn) gde je m broj upita, a n dužina niza što nije efikasno kod velikih nizova ili velikog broja upita.
- Drugo rešenje jeste da umesto čuvanja elemenata niza čuvamo zbirove prefiksa tako da zbir P_i predstavlja zbir prvih i elemenata. Zbir segmenta [a, b] onda dobijamo kao razliku zbira prefiksa do elementa b i zbira prefiksa do elementa a - 1: P_{b+1} - P_a. Prefiksni zbirovi se izračunavaju u vremenu O(n) i smeštaju u pomoćni ili originalni niz, a zbir svakog segmenta se potom računa u konstantnoj složenosti. Ukupna složenost je dakle O(m + n).

Problem: Efikasno menjati segmente [a, b] nekog niza tako da svi elementi segmenta budu uvećani za neku vrednost c.

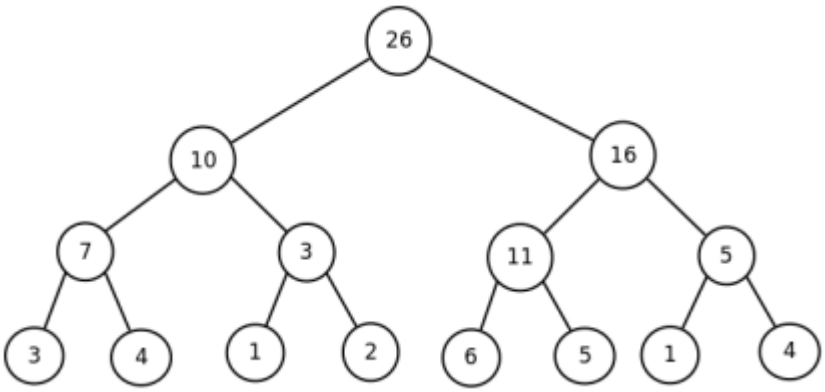
- Rešenje grubom silom bi uvećavalo sve elemente segmenta pa bi za m upita i n članova niza složenost opet bila O(mn).
- Drugo rešenje jeste da umesto čuvanja elemenata niza čuvamo razlike susednih elemenata: $R_0 = x_0$, $R_i = x_i - x_{i-1}$. Tokom uvećanja svih elemenata segmenta [a, b] za neku vrednost c menjaju se samo razlike između elemenata na pozicijama a i a - 1 kao i između elemenata na pozicijama b + 1 i b. Razlike susednih elemenata se izračunavaju u složenosti O(n), a takođe i originalni niz od niza razlika možemo dobiti u vremenu O(n) izračunavanjem zbira prefiksa niza razlika. Tu uočavamo povezanost ove dve tehnike.

Primećujemo da su kod ovih **statičkih upita nad rasponima** zbirovi prefiksa i razlike susednih elemenata davale efikasnije rešenja, međutim ako imamo **dinamičke upite nad rasponima** gde želimo istovremeno efikasno da vršimo obe vrste upita onda dolazimo do problema. Zbirovi prefiksa omogućavaju efikasno postavljanje upita nad segmentima ali ažuriranje elemenata nije efikasno jer je potrebno ažurirati sve zbirove prefiksa nakon jednog ažuriranog elementa. Sa druge strane, niz razlika susednih elemenata omogućava efikasno ažuriranje niza ali izvršavanje upita očitavanja stanja niza podrazumeva rekonstrukciju polaznog niza što opet nije efikasno. Efikasno rešavanje obe vrste upita daju dve strukture podataka - **segmentna drveta** i **Fenvikova drveta**.

3. Segmentna drveta - pojam, svojstva, primeri, formiranje segmentnog drveta

Segmentno drvo (segment tree) je struktura podataka koja omogućava efikasno rešavanje upita raspona. Segmentna drveta se mogu koristiti i za operaciju sabiranja, ali i za druge asocijativne operacije kao što su određivanje najvećeg ili najmanjeg elementa, NZD-a, NZS-a i slično. Elementi niza predstavljaju listove drveta. Oni se grupišu dva po dva i računa se njihov zbir (ili proizvod, min, max, NZD, NZS). Ukoliko broj elemenata niza nije stepen broja 2 možemo ga dopuniti nulama (zbir ili NZD) ili jedinicama (proizvod, NZS). Dobijeni nivo ponovo grupišemo u parove i računamo sledeći nivo, sve dok ne dođemo do čvora koji će imati samo jedan nivo. Ovu strukturu je najlakše implementirati preko niza, tako što se koren nalazi na poziciji 1. Polazni elementi niza će se nalaziti na pozicijama
[n, 2n - 1] (element koji se nalazio na poziciji p biće na poziciji p + n). Levo dete čvora k će biti na poziciji 2k, a desno na poziciji 2k + 1. Roditelj čvora k nalazi se na poziciji $\lfloor \frac{k}{2} \rfloor$.

Primer: segmentno drvo za sabiranje niza [3, 4, 1, 2, 6, 5, 1, 4]:



(na ovom nizu i njegovom drvetu su prikazani primeri za sva pitanja iz segmentnih drveta)

Formiranje segmentnog drveta:

- **naviše**: Prvo se popunjavaju listovi, pa onda unutrašnji čvorovi sve dok se ne dođe do korena. Elementi polaznog niza se prekopiraju u niz koji predstavlja drvo, počevši od pozicije n. Zatim se svi ostali čvorovi drveta (od pozicije n - 1, do pozicije 1) popunjavaju kao zbirovi svoje dece - na poziciju k upisujemo zbir elemenata na pozicijama 2k i 2k + 1.
- **naniže**: Ova implementacija je malo neefikasnija, ali u nekim situacijama je neophodna. Drvo se popunjava rekurzivno tako što popunimo levo poddrvo i desno poddrvo pa koren popunimo kao zbir ta dva. Ako koren pokriva zbir segmenta [x, y] onda levo dete pokriva segment [x, $\lfloor \frac{x+y}{2} \rfloor$], a desno dete pokriva segment [$\lfloor \frac{x+y}{2} \rfloor$ + 1, y]. Izlaz iz rekurzije predstavljaju listovi koje prepoznajemo po tome što pokrivaju segment dužine 1 i u njih samo kopiramo elemente početnog niza.

4. Segmentna drveta - računanje zbira elemenata iz nekog segmenta, primeri

Izračunavanje odozdo naviše: Za sve unutrašnje elemente segmenta čiji zbir računamo znamo da im se zbir nalazi u čvorovima iznad njih. Izuzetak mogu biti samo elementi na krajevima segmenta. Ako je element na levom kraju segmenta levo dete (parna pozicija), onda se njegov zbir nalazi u roditeljskom čvoru. Ako je desno dete (neparna pozicija) onda njegov roditeljski čvor sadrži zbir sa elementom koji ne pripada segmentu, pa ćemo ovaj element posebno dodati u zbir umesto njegovog roditelja. Ako je element na desnom kraju segmenta levo dete

(parna pozicija), onda njegov roditelj sadrži zbir sa elementom koji nije deo segmenta pa ćemo element posebno dodati. Ako je desno dete (neparna pozicija) tada se njegov zbir nalazi u roditeljskom čvoru.

- **Primer**: Računanje zbira segmenta [2, 6] niza od 8 elemenata iz prethodnog pitanja. Elementi niza na pozicijama od 2 do 6 se u drvetu nalaze na pozicijama 10 do 14. Leva granica je parna, ne radimo ništa. Desna granica je parna pa njen roditelj sadrži zbir sa elementom koji ne pripada traženom segmentu. Zbog toga na zbir dodajemo element na poziciji 14, pa je trenutni zbir sada jednak 1, a granicu spuštamo na 13. Granice delimo sa 2 i prelazimo na roditeljski nivo. Sada posmatramo pozicije [5, 6]. Leva granica je neparna pa na zbir dodajemo njen zbir (3) i povećamo granicu na 6. Desna granica je parna pa na zbir dodajemo njen zbir (11) i smanjimo granicu na 5. Granice su se mimoišle, došli smo do kraja i zbir je 1 + 3 + 11 = 15, što odgovara zbiru elemenata 1 + 2 + 6 + 5 + 1.

Izračunavanje odozgo naniže: Izračunavanje kreće od korena i u svakom koraku poredimo trenutni segment sa segmentom čiji zbir tražimo. Postoje tri različita moguća odnosa između njih. Ako su disjunktni doprinos tekućeg čvora je nula. Ako je tekući segment potpuno sadržan u traženom onda je doprinos tekućeg čvora potpun. Ako se segmenti seku doprinos tekućeg čvora jednak je zbiru doprinosa njegovog levog i desno deteta.

- **Primer**: Računanje zbira segmenta [2, 6] niza iz prethodnog pitanja. Izvršavanje kreće od korena. Segment [0, 7] se seče sa segmentom [2, 6] te će zbir biti jednak sumi doprinosa segmenata [0, 3] i [4, 7]. Segment [0, 3] se seče sa segmentom [2, 6] te opet startujemo dva rekurzivna poziva za segmente [0, 1] i [2, 3]. Segment [0, 1] je disjunktan sa segmentom [2, 6] pa je njegov doprinos traženoj sumi 0, a segment [2, 3] je sadržan u segmentu [2, 6] te je njegov doprinos potpun - jednak je vrednosti 3 koju taj čvor čuva. S druge strane, segment [4, 7] se seče sa segmentom [2, 6] te opet startujemo dva rekurzivna poziva za segmente [4, 5] i [6, 7]. Segment [4, 5] je u potpunosti sadržan u segmentu [2, 6] te je njegov doprinos potpun i iznosi 11, a segment [6, 7] se seče sa segmentom [2, 6], pa iz njega startujemo dva rekurzivna poziva: za segmente [6, 6] i [7, 7]: prvi je u potpunosti sadržan u traženom segmentu, te je njegov doprinos potpun i iznosi 1, a drugi je disjunktan pa je njegov doprinos jednak nuli. Dakle, ukupna vrednost sume segmenta biće jednaka 3 + 11 + 1 = 15.

5. Segmentna drveta - ažuriranje vrednosti elementa, primeri

Ažuriranje odozdo naviše: Prilikom ažuriranja vrednost nekog elementa, potrebno je ažurirati sve čvorove na putanji od tog lista do korena. S obzirom da znamo da se roditeljski čvor čvora k nalazi na poziciji $\lfloor \frac{k}{2} \rfloor$ ova operacija se lako implementira.

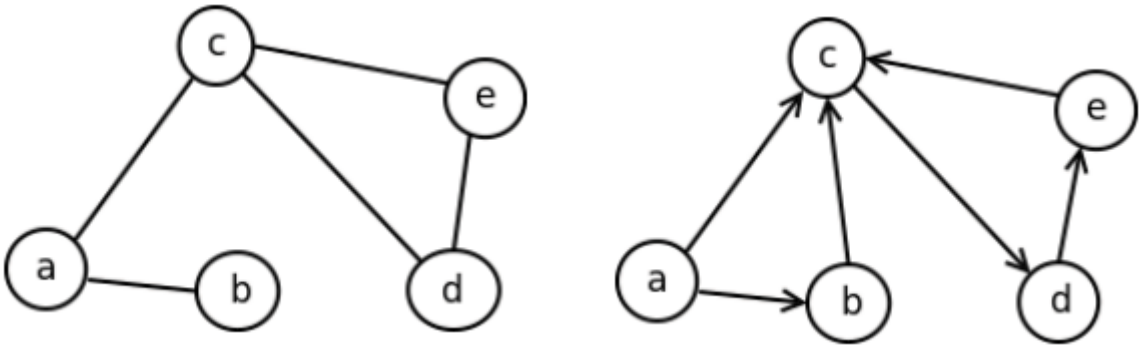
- **Primer**: Ažuriranje elementa na poziciji 2 niza iz prethodnog pitanja. Element na poziciji 2 će se nalaziti na poziciji 2 + 8 = 10 u drvetu. Nakon što smo ga ažurirali, ažuriramo njegovog roditelja $\lfloor \frac{10}{2} \rfloor$ = 5, zatim njegovog roditelja $\lfloor \frac{5}{2} \rfloor$ = 2, zatim njegovog roditelja $\lfloor \frac{2}{2} \rfloor$ = 1. Čvor na poziciji 1 je koren pa se tu zaustavljamo.

Ažuriranje odozgo naniže: U ovoj implementaciji polazimo od korena i u svakom koraku proveravamo da li se element koji treba ažurirati nalazi u levom ili desnom poddrvetu. Rekurzivno se spuštamo u poddrvo u kom je taj element i ponavljamo postupak sve dok ne dođemo do lista koji ažuriramo. Prilikom povratka na gore ažuriramo vrednosti svih čvorova kroz koje smo prošli tako što ponovo saberemo njegovo levo i desno dete (čvorovi na pozicijama 2k i 2k + 1).

- **Primer**: Ažuriranje elementa na poziciji 2 niza iz prethodnog pitanja. Krećemo od korena koji pokriva segment [0, 7]. Element na poziciji 2 će se nalaziti u poddrvetu koje pokriva segment [0, 3] pa se spuštamo u njega. Na isti način se spuštamo u čvor koji pokriva segment [2, 3]. Na kraju, spuštamo se u segment [2, 2] i pošto smo došli u list ažuriramo vrednost elementa. Pri povratku iz rekurzije ažuriramo i čvorove koji pokrivaju segmente [2, 3], [0, 3] i [0, 7].

6. Grafovi - osnovni pojmovi, reprezentacije netežinskih i težinskih grafova

Graf G = (V, E) sastoji se od skupa **čvorova** (vertex) V i skupa **grana** (edge) E. Grana najčešće odgovara paru različitih čvorova, ali nekad su dozvoljene i **petlje** - grane koje povezuju čvor sa samim sobom. Graf može biti **usmeren** ili **neusmeren**. Kod usmerenih grafova je bitan redosled čvorova kod grana. Primer neusmerenog (levo) i usmerenog grafa (desno):



Susedni čvor čvora v je svaki čvor u do kog postoji grana iz čvora v. **Stepen čvora** v - d(v) je broj susednih čvorova čvora v. Kod usmerenih grafova razlikujemo **ulazni stepen** čvora v koji je jednak broju grana za koje je čvor v kraj i **izlazni stepen** čvora v koji je jednak broju grana za koje je čvor v početak.

Put od čvora v_1 do čvora v_k je niz čvorova v_1, v_2, . . ., v_k povezanih granama (v_1, v_2), (v_2, v_3), . . ., (v_{k-1}, v_k). **Prost put** je put u kom se svaki čvor pojavljuje samo jednom. Čvor u je **dostižan** iz čvora v ako postoji put od čvora v do čvora u. **Ciklus** je put čiji se prvi i poslednji čvor poklapaju. **Prost ciklus** je ciklus ako se, sem prvog i poslednjeg čvora, ni jedan drugi čvor ne javlja dva puta.

Neusmereni oblik usmerenog grafa je isti graf, bez smerova na granama. Neusmeren graf je **povezan** ako postoji put između svaka dva čvora u grafu. Usmereni graf je **slabo povezan** ako u njegovom neusmerenom obliku postoji put između svaka dva čvora u grafu. Usmeren graf je **jako povezan** ako za svaka dva čvora u njemu postoji usmeren put. Neusmereni graf je **šuma** ako ne sadrži cikluse. **Drvo** je povezana šuma.

Graf H = (U, F) je **podgraf** grafa G = (V, E) ako važi U $\subseteq$ V i F $\subseteq$ E. **Povezujuće drvo** neusmerenog grafa je njegov podgraf koji je drvo i sadrži sve njegove čvorove. **Povezujuća šuma** neusmerenog grafa je njegov podgrad koji je šuma i sadrži sve njegove čvorove. Ako neusmereni graf nije povezan, onda se on može na jedinstven način razložiti u skup povezanih podgrafova koji se nazivaju **komponente povezanosti** grafa.

Grafove možemo predstaviti na dva uobičajena načina:

- **Matrica povezanosti (susedstva)** je kvadratna matrica reda n, gde je n broj čvorova grafa. Element a_{ij} će imati vrednost 1 ako postoji grana (v_i, v_j), odnosno 0 ako ne postoji. Ako je graf neusmeren matrica će biti simetrična. Nedostatak ovog načina reprezentacije je to što matrica uvek zauzima prostor veličine n^2 čak i kada je broj grana grafa jako mali. Složenost operacija dodavanja ili uklanjanja grane iz grafa, kao i ispitivanja da li neka grana postoji je O(1). Prolazak kroz sve susedne čvorove nekog čvora je O(n).
- **Lista povezanosti (susedstva)** podrazumeva da se svakom čvoru pridružuje povezana lista koja sadrži sve čvorove do kojih postoji grana iz tog čvora. Dakle, graf je predstavljen nizom lista. Svaki element niza sadrži indeks čvora i pokazivač na njegovu listu suseda. Za implementaciju se ne moraju koristiti samo liste, moguće je koristiti i dinamički niz, balansirana binarna drveta ili heš-tabele. Ako je graf statički može se koristiti i statički niz dužine |V| + |E| u slučaju usmerenog, odnosno |V| + 2|E| u slučaju neusmerenog grafa. Na poziciji 0 niza nalazi se broj čvorova grafa, dok je narednih |V| pozicija pridruženo čvorovima i one sadrže indeks početka spiska čvorova susednih tom čvoru. Liste povezanosti su prostorno efikasnije - O(|V| + |E|).

Operacije ispitivanja da li grana pripada grafu kao i uklanjanje grane iz grafa su u najgorem slučaju složenosti O(|V|) ili O(1) kada se koriste heš-tabele. Dodavanje novog čvora u graf je jednostavnije nego kod matrica povezanosti. Prolazak kroz sve susede čvora v je složenosti O(1 + d(v)).

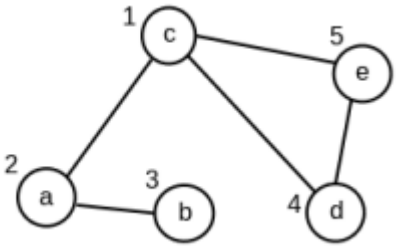
Težinski graf je graf u kom je svakoj grani pridružena **težina**. **Dužina puta** je zbir svih težina grana na tom putu, i tada težine često nazivamo **dužine**. U slučaju težinskih grafova matrice povezanosti umesto logičkih vrednosti mogu da sadrže težine grana i neku specijalnu vrednost ako grana ne postoji. U slučaju listi povezanosti elementi listi će pored susednih čvorova sadržati i težinu grane do tog čvora.

7. Grafovi - algoritam pretrage u dubinu, primer, ulazna i izlazna obrada, složenost DFS pretrage, dokaz da DFS obilazi ceo neusmeren povezan graf, određivanje komponenti povezanosti neusmerenog grafa

Pretraga u dubinu (DFS: depth-first-search) podrazumeva obilazak grafa tako da uvek kada je to moguće idemo dalje u dubinu pre nego što se vratimo u čvor u kom smo već bili. Korist ove pretrage je jednostavnost i laka realizacija. Ako je graf dat listom povezanosti algoritam je sledeći:

- Na početku su svi čvorovi neoznačeni, a označavaćemo ih kada ih po prvi put posetimo. Čvor r iz koga se pokreće pretraga označava se kao posećen. U listi suseda čvora r pronalazimo prvi neoznačeni sused r_1 i iz njega rekurzivno pokrećemo DFS. Kada se završi rekurzivni poziv za čvor r_1 prelazi se na sledeći neoznačeni sused čvora r itd. Dakle, iz rekurzije za čvor r izlazimo ako su mu svi susedi već označeni.

Primer: Na sledećem grafu je DFS pokrenut iz čvora c:



- Označavamo čvor c. U njegovoj listi suseda nalaze se čvorovi a, d i e. Rekurzivno prozivamo pretragu iz čvora a pošto je neoznačen. Označavamo čvor a. Njegov sused je b pa pokrećemo DFS iz čvora b. Označavamo čvor b. On nema neoznačenih suseda pa se vraćamo u čvor a. Čvor a nema više neoznačenih suseda pa se vraćamo u čvor c. Čvor c ima neoznačene susede d i e, pa pokrećemo DFS iz d. Označavamo čvor d. Čvor d ima neoznačenog suseda e pa pokrećemo DFS iz čvora e. Označavamo čvor e. Čvor e nema neoznačenih suseda pa se vraćamo u čvor d. Čvor d nema više neoznačenih suseda pa se vraćamo u čvor c. Čvor c nema više neoznačenih suseda i tu je kraj pretrage. Dakle čvorovi su obiđeni u redosledu c, a, b, d, e.

Pretraga grafa se uvek vrši sa nekim ciljem. Posetama čvora ili grane mogu se pridružiti dve vrste obrade. **Ulazna obrada** vrši se u trenutku označavanja čvora, odnosno pre pokretanja DFS pretrage za potomke datog čvora. **Izlazna obrada** vrši se na kraju, kada obradimo sve potomke datog čvora.

Lema: Ako je graf povezan, onda su po završetku pretrage u dubinu svi čvorovi označeni, a sve grane pregledane bar po jednom.

- **Dokaz**: Pretpostavimo suprotno i sa U označimo skup neoznačenih čvorova nakon pretrage. Pošto je graf povezan bar jedan čvor skupa U mora biti povezan granom sa nekim čvorom w koji je označen. Ovde

dolazimo do kontradikcije jer ako je w označen onda su i svi njegovi susedi označeni. Pošto su svi čvorovi posećeni, a kad se čvor poseti pregledaju se sve grane koje vode iz njega, zaključujemo da su i sve grane grafa pregledane. □

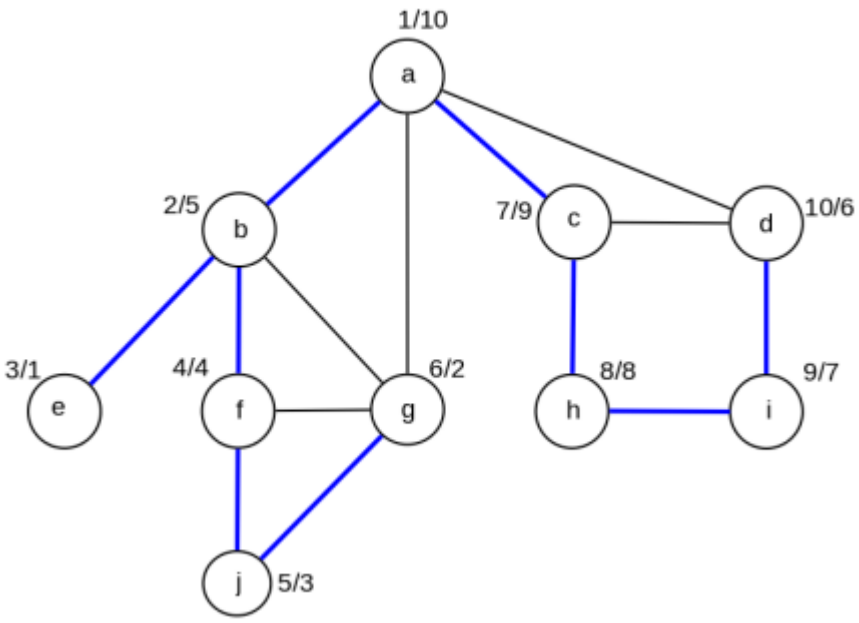
Složenost: Prilikom izvršavanja DFS algoritma na neusmerenom grafu koji je zadat listom povezanosti, svaka grana se pregleda tačno dva puta, po jednom sa svakog kraja. Dakle svi rekurzivni pozivi se ukupno izvršavaju u vremenu O(|E|). Broj rekurzivnih poziva je |V|, pa je ukupa složenost DFS algoritma O(|V| + |E|). Ukoliko se za reprezentaciju koristi matrica povezanosti onda je za prolazak kroz susede jednog čvora potrebno vreme O(|V|), pa je ukupna složenost O($|V|^2$). Zaključujemo da je za DFS algoritam efikasnije koristiti liste povezanosti za reprezentaciju grafa.

DFS se može prilagoditi tako da radi i za nepovezane grafove. Posle prvog pokretanja DFS algoritma, možemo proveriti da li su svi čvorovi označeni. Ako nisu, možemo ponovo pokrenuti pretragu polazeći od nekog od neoznačenih čvorova i tako sve dok ne označimo sve čvorove. Na ovaj način možemo lako proveriti da li je graf povezan ili ne i izačunati **broj komponenti povezanosti** ako nije. Graf će imati onoliko komponenti povezanosti koliko puta smo morali da pokrenemo DFS ispočetka.

8. Grafovi - konstrukcija DFS drveta i DFS numeracija, klasifikacija grana neusmerenog grafa, primer

Prilikom obilaska grafa u dubinu, možemo izdvojiti sve grane preko kojih idemo ka do tada neoznačenim čvorovima. Preko tih grana se dostižu svi čvorovi pa je podgraf koji čine te grane povezan. On nema cikluse, jer se od svih grana koje vode u neki čvor izdvaja samo jedna. Dakle u slučaju povezanog grafa ovaj podgraf je povezujuće drvo koje nazivamo **DFS drvo** grafa. Čvor iz koga se startuje obilazak biće koren DFS drveta. **Dolazna DFS numeracija** (**preOrder**) podrazumeva numerisanje čvorova prema redosledu označavanja, tj. čvorovi se numerišu kada prvi put stignemo u njih. **Odlazna DFS numeracija** (**postOrder**) podrazumeva numerisanje čvorova prema redosledu napuštanja čvorova. Konstrukcija DFS drveta i DFS numeracija predstavljaju dve veoma važne primene DFS algoritma.

Primer: Na sledećem grafu su označene DFS numeracije (preOrder/postOrder), kao i DFS drvo datog grafa (plava boja) ako se DFS pokrene iz čvora a.



Čvor v je **predak** čvora w u DFS drvetu sa korenom r ako se v nalazi na jedinstvenom putu od r do w u tom drvetu. Tada je čvor w **potomak** čvora v. DFS pretraga će pre biti pokrenuta za pretke nekog čvora nego za njega samog, pa važi v.Pre < w.Pre, ali će se završiti posle pa važi v.Post > w.Post. Redosled sinova nekog čvora dat je listom povezanosti, pa se može reći koji je od dva sina levi, a koji desni. Relacija levi-desni se može preneti i na proizvoljna dva čvora. Za dva čvora u i v postoji jedinstveni zajednički predak w u DFS drvetu, kao i sinovi čvora w - u' (predak od u) i v' (predak od v). Tada je čvor u levo od čvora v akko je čvor u' levo od čvora v'. Jasno je da onda važi u.Pre < v.Pre i u.Post < v.Post jer se ranije stiže u čvor v i pre se izlazi iz čvora v.

Osnovna osobina DFS drveta neusmerenog grafa: Neka je G = (V, E) povezan neusmeren graf i neka je T = (V, F) njegovo DFS drvo. Svaka grana grafa ili pripada drvetu T ili spaja dva čvora grafa G od kojih je jedan predak drugog u drvetu T.

- **Dokaz**: Neka je (v, u) grana grafa i pretpostavimo da je tokom DFS pretrage čvor v posećen pre čvora u. Kada je označen čvor v, pokreće se DFS iz svih njegovih susednih čvorova. Kada dođe red na čvor u on može biti već označen i u tom slučaju je u potomak čvora v u drvetu T. Ako nije označen onda se pokreće DFS iz njega i on postaje sin čvora v u drvetu T. □

Tvrđenje može i da glasi: grane grafa ne mogu biti **poprečne grane** u odnosu na DFS drvo, tj. ne mogu da povezuju dva čvora u odnosu levo-desno. Na primeru grafa sa slike, grane (a, g), (b, g), (a, d), (c, d) i (f, g) su grane koje spajaju pretke i potomke DFS drveta, a ostale grane su grane DFS drveta.

9. Grafovi - klasifikacija grana usmerenog grafa, primer, odnos odlazne numeracije i tipa grane

DFS algoritam u slučaju usmerenih grafova isti je kao i za neusmerene grafove ali usmerena DFS drveta imaju drugačije osobine od neusmerenih. U opštem slučaju DFS pretraga će dati DFS šumu, ali ako je usmereni graf jako povezan onda će dati DFS drvo. U odnosu na DFS drvo grane grafa mogu biti:

- **grane DFS drveta** - povezuju oca sa sinom
- **povratne grane** - povezuju potomka sa pretkom
- **direktne grane** - povezuju pretka sa potomkom
- **poprečne grane** - povezuju čvorove koji su u relaciji levo-desno

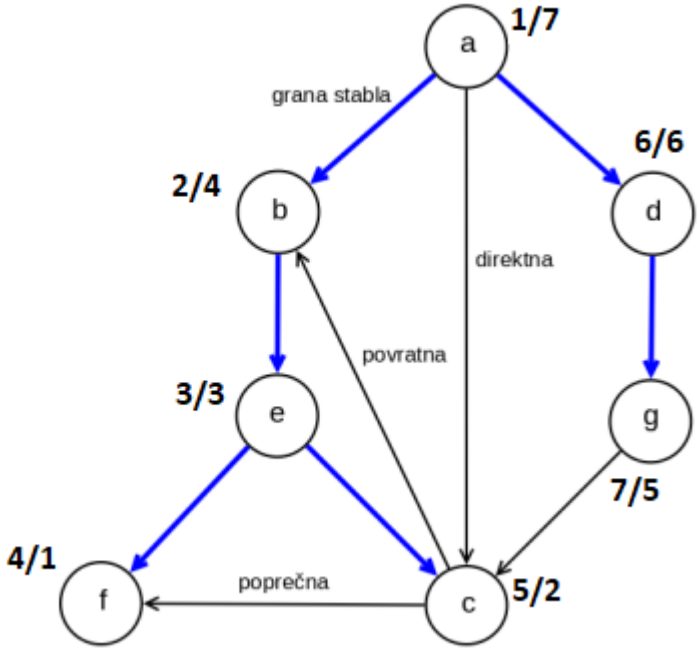
Osnovna osobina DFS drveta usmerenog grafa: Neka je G = (V, E) usmeren graf i neka je T = (V, F) njegovo DFS drvo. Ako je (v, w) grana grafa za koju važi v.Pre < w.Pre, onda je čvor w potomak čvora v u drvetu T.

- **Dokaz**: Iz v.Pre < w.Pre zaključujemo da je čvor w označen posle čvora v pa grana (v, w) mora biti razmatrana u toku DFS pretrage iz čvora v. Ako u tom trenutku w nije označen, onda se grana (v, w) uključuje u DFS drvo pa lema važi. Ako je w već bio označen onda je to urađeno tokom izvođenja DFS pretrage iz čvora v, pa je w potomak čvora v u drvetu T.

Za usmerenu granu (u, v) važi:

- u.Post $\leq$ v.Post $\Rightarrow$ grana je **povratna**
- u.Post > v.Post i u.Pre > v.Pre $\Rightarrow$ grana je **poprečna**
- u.Post > v.Post i u.Pre < v.Pre $\Rightarrow$ grana je **grana DFS drveta** ako je čvor u roditelj čvora v u DFS drvetu, a inače **direktna**

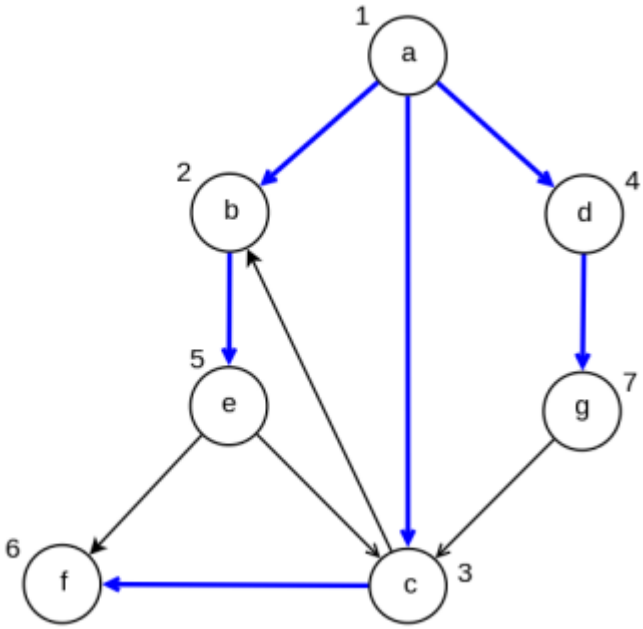
Primer: grane DFS drveta su označene plavom bojom, direktna grana je (a, c), poprečne grane su (c, f) i (g, c), povratna grana je (c, b).



10. Grafovi - algoritam pretrage u širinu, primer, složenost BFS pretrage, BFS drvo, BFS numeracija

Pretraga u širinu (BFS: breadth-first-search) je obilazak grafa nivo po nivo, pri čemu se usput formira **drvo pretrage u širinu (BFS drvo)**. BFS drvo se formira uključivanjem samo grana ka neoznačenim čvorovima. Takav graf neće imati ciklusa jer se od svih grana ka čvoru uzima samo jedna pa će zaista biti drvo. Ako je graf zadat listom povezanosti i krenemo od čvora v, onda se najpre posećuju svi susedi čvora v u redosledu određenom njegovom listom suseda i oni će biti sinovi čvora v u BFS drvetu. To su čvorovi nivoa 1. Zatim se dolazi do svih njihovih neposećenih suseda i oni će biti unuci čvora v, tj. čvorovi nivoa 2 itd. Kraj pretrage je kada više nijedan čvor na nekom nivou nema neoznačenih suseda. Pri obilasku čvorovi se mogu numerisati **BFS brojevima**. Čvor će imati BFS broj k ako je k-ti po redu čvor označen u toku BFS pretrage. Ovde izlazna obrada nema smisla je se ide samo naniže, odnosno nema povratka naviše.

Primer:



- BFS pretragu pokrećemo iz čvora a. Redom posećujemo njegove susede b, c i d. Zatim posećujemo neoznačene susede čvora b - čvor e, neoznačene susede čvora c - čvor f i neoznačene susede čvora d - čvor g. Čvorovi e, f i g nemaju neoznačenih suseda pa je tu kraj BFS pretrage. Čvorovi su obiđeni u redosledu a, b, c, d, e, f, g.

Složenost algoritma je O(|V| + |E|) jer se svaki čvor obrađuje po jednom, a svaka grana se pregleda jednom u slučaju usmerenog odnosno dva puta u slučaju neusmerenog grafa. Slično kao i kod DFS pretrage, složenost će biti $O(n^2)$ ako se za reprezentaciju koristi matrica povezanosti.

Leme:

- Ako grana (u, v) pripada BFS drvetu grafa i čvor u je otac čvora v onda čvor u ima najmanji BFS broj među čvorovima iz kojih postoji grana ka v.
- Put od korena r BFS drveta do proizvoljnog čvora w kroz BFS drvo najkraći je put od čvora r do čvora w u grafu G. Dužina najkraćeg puta između dva čvora, odnosno **rastojanje** d(r, w), je broj grana tog puta. To rastojanje odgovara **nivou čvora** u odnosu na BFS drvo.
- Ako je graf neusmeren i (v, w) neka njegova proizvoljna grana, onda ta grana spaja dva čvora čiji se nivoi razlikuju najviše za jedan.

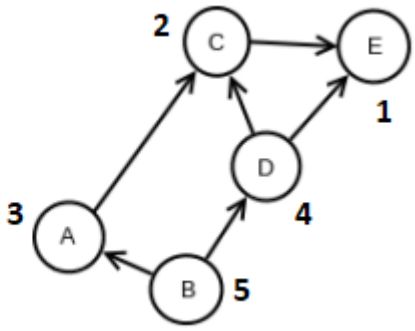
11. Grafovi - topološko sortiranje - algoritam po izboru, primer

Ukoliko imamo više operacija ili akcija koje treba izvršiti u određenom redosledu i gde te akcije međusobno zavise ovom problemu možemo pridružiti graf i onda problem određivanja redosleda akcija nazivamo **topološko sortiranje grafa**. Svakom poslu možemo pridružiti čvor, a zavisnosti prikazati pomoću usmerenih grana tako da grana (x, y) označava da posao y ne može biti započet pre završetka posla x. Potrebno je odrediti numeraciju čvorova tako da za sve grane važi da da je polazni čvor numerisan manjom vrednošću nego završni čvor. Graf mora biti bez usmerenih ciklusa jer u protivnom neki poslovi ne bi nikada bili započeti.

U usmerenom acikličkom grafu ne postoji ciklus, pa samim tim ne postoje ni povratne grane u odnosu na DFS drvo. To znači da za svaku granu (u, v) važi u.Post > v.Post. Vidimo da ako čvorove grafa uredimo u opadajućem redosledu u odnosu na odlaznu numeraciju dobićemo topološko uređenje grafa. Algoritam se dakle svodi na DFS

pretragu i označavanje čvorova pri izlaznoj obradi. Na kraju se redosled obrne i dobijamo topološko uređenje. Složenost sortiranja odgovara složenosti DFS pretrage i biće O(|E| + |V|).

Primer:

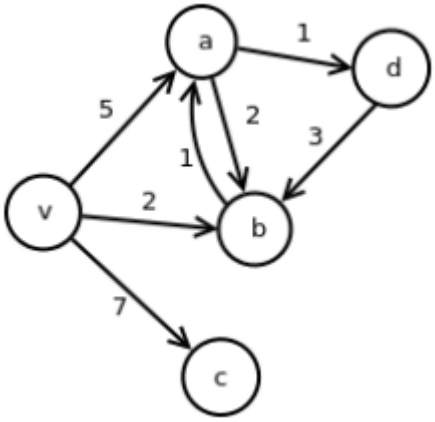


- DFS pretragu možemo pokrenuti iz čvora B. Odlazna numeracija će dati redosled E, C, A, D, B. Kada obrnemo redosled dobijamo poredak B, D, A, C, E što odgovara traženom topološkom uređenju.

12. Grafovi - Dajkstrin algoritam za određivanje najkraćih puteva iz zadatog čvora, primer

Dajkstrin algoritam se može primeniti na grafove koji mogu imati i cikluse, ali se pretpostavlja da su dužine svih grana pozitivne. Zasniva se na induktivnoj hipotezi da za čvor v umemo da nađemo k njemu najbližih čvorova, kao i dužine najkraćih puteva do njih. Prvi najbliži čvor je čvor do koga ide grana najmanje dužine od čvora v. Sledeći najbliži čvor može biti ili neki od preostalih suseda čvora v (onaj sa granom najmanje dužine) ili neki čvor do koga se od v može doći preko čvora koji smo odredili kao najbliži čvoru v u prvom koraku. Ako sa V_k označimo skup k najbližih čvorova čvoru v, želimo da u sledećem koraku nađemo sledeći najbliži čvor i da njega ubacimo u skup. Jasno je da treba proveravati samo grane koje spajaju neki čvor iz skupa V_k sa čvorovima van tog skupa, najkraći put do sledećeg najbližeg čvora w ne može da sadrži još neki čvor van V_k jer onda w ne bi bio najbliži. Dakle, u svakoj iteraciji dodajemo novi čvor u skup V_k, čvor koji će dati najmanji zbir trenutnog puta od čvora v do nekog čvora u iz V_k i grane od tog čvora u do čvora w van skupa. Dodatno, ne moramo u svakom koraku proveravati sve vrednosti u.SP + d(u, w). Možemo pamtiti dužine trenutno poznatih najkraćih puteva do svih čvorova van V_k i da im popravljamo vrednosti prilikom proširivanja skupa. Jedini način da se smanji dužina puta je ako novi najkraći put prolazi kroz dodati čvor w. Za svaki čvor z van skupa onda proveravamo da li je w.SP + d(w, z) < z.SP i ažuriramo vrednost ako jeste. Drvo koje se formira određivanjem svih najkraćih puteva od nekog čvora do ostalih čvorova naziva se **drvo najkraćih puteva**. Za efikasno pronalaženje najmanjeg elementa možemo korisiti min-hip, tj. red sa prioritetom.

Primer: odrediti sve najkraće puteve od čvora v.



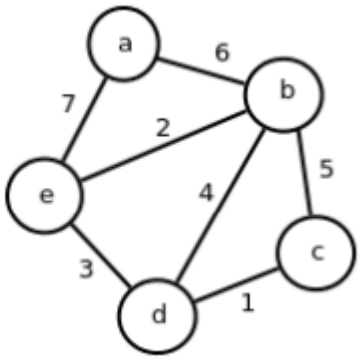
- Čvor v dodajemo u red. Dužina najkraćeg puta do njega je 0, a do ostalih čvorova $+\infty$.
- Uzimamo ga iz reda, prolazimo kroz njegove susede i proveravamo da li možemo popraviti neke puteve. Novi najkraći put do čvorova a, b i c će biti direktne grane od čvora v do njih, dužina 5, 2 i 7. Ažuriramo im najkraće puteve i dodajemo ih u red.
- Sledeći najbliži čvor je čvor b jer je put do njega dužine 2. Uzimamo ga iz reda i opet proveravamo da li preko njega možemo skratiti neki put. Imamo granu ka čvoru a pa najkraću dužinu puta do a ažuriramo na 2 + 1 = 3.
- Sledeći najbliži čvor je a jer je put do njega dužine 3. Skidamo čvor a sa reda. Od a imamo granu ka čvoru d pa najkraću dužinu puta do d ažuriramo na 3 + 1 = 4. Stavljamo čvor d u red.
- Sledeći najbliži čvor je d jer je put do njega dužine 4, skidamo ga sa reda. Preko njega ne možemo smanjiti nijedan najkraći put.
- Sledeći najbliži čvor je c jer je put do njega dužine 7, skidamo ga sa reda. Od čvora c nemamo nijednu granu.
- Red je prazan tako da su svi najkraći putevi određeni. Najbliži čvor je b, zatim a, zatim d i na kraju c.
- Možemo da odredimo i putanje tako što gledamo gde se desila promena. Najkraći put od v do b je (v, b). Najkraći put od v do a je (v, b, a) jer se najkraći put promenio dodavanjem b u red. Najkraći put od v do d je (v, b, a, d) jer se najkraći put promenio dodavanjem čvora a u red, a za čvor a se promenio dodavanjem b u red. Najkraći put od v do c je (v, c).

13. Grafovi - algoritam po izboru za određivanje minimalnog povezujućeg drveta, primer

Minimalno povezujuće drvo je povezani podgraf grafa koji sadrži sve čvorove grafa i zbir dužina grana mu je minimalan.

Primov algoritam se bazira na indukciji po broju izabranih grana. Induktivna hipoteza: za dati graf umemo da pronađemo povezan podgraf - drvo T sa k grana, k < |V| - 1, tako da je drvo T podgraf minimalnog povezujućeg drveta grafa. Bazni slučaj je kada biramo prvu granu i on je trivijalan - uzimamo granu minimalne težine u grafu. Pretpostavimo da smo pronašli drvo T i da sada treba da izaberemo sledeću granu. Pošto znamo da je T podgraf traženog minimalnog povezujućeg drveta, to znači da mora postojati grana koja povezuje T sa nekim čvorom koji nije deo T. Najmanja takva grana će pripadati minimalnom povezujućem drvetu. Označimo tu granu sa (u, v) gde u pripada T, a v ne. Ako ona ne pripada minimalnom povezujućem drvetu onda postoji neki drugi put do u do v preko neko grane (x, y) gde x pripada T, a y ne. Međutim ta grana je duža od grane (u, v) pa samim tim minimalno povezujuće drvo ne bi bilo minimalno. Sledi da (u, v) mora biti deo rezultujućeg drveta. Ako u trenutku razmatranja naiđemo na dve grane iste dužine uzimamo bilo koju od njih. Primov algoritam je veoma sličan Dajkstrinom algoritmu, samo se minimum traži po dužinama grana, a ne dužinama puta.

Primer:



- U drvo dodajemo granu (d, c) jer je najmanje težine. Razmatramo grane čiji je jedan čvor čvor d ili čvor c, a drugi čvor je van skupa tih čvorova. To su grane (d, e), (d, b) i (c,b).
- U drvo dodajemo granu (d, e) jer je najmanje težine. Razmatramo samo grane koje na jednom kraju imaju čvorove d, c ili e, a na drugom neki čvor van tog skupa. To su grane (d, b), (c, b), (e, b) i (e, a).
- U drvo dodajemo granu (e, b) jer je najmanje težine. Razmatramo grane (e, a) i (b, a).
- U drvo dodajemo granu (b, a) jer je najmanje težine. Svi čvorovi su dodati - kraj algoritma.

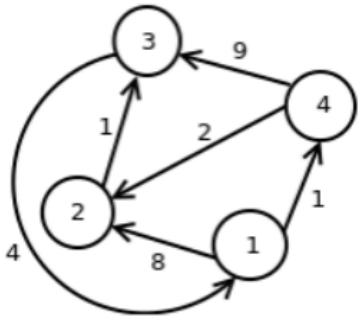
14. Grafovi - Flojd-Varšalov algoritam za određivanje svih najkraćih puteva u grafu, primer

Flojd-Varšalov algoritam se zasniva na uvođenju ograničenja na tip dozvoljenih puteva. Podrazumeva se da u grafu nema ciklusa negativne dužine. Indukcija se izvodi po opadajućem broju ograničenja tako da na kraju dolaze u obzir svi putevi odnosno ima 0 ograničenja. **K-put** između čvorova u i v jer put koji čine samo čvorovi čiji je redni broj manji ili jednak od k. 0-put od čvora u do čvora v jer put koji se sastoji samo od direktne grane između njih. Induktivna hipoteza je da umemo da odredimo dužine najkraćih (m - 1)-puteva između svaka dva čvora. Bazni slučaj za m = 1 razmatra samo 0-puteve odnosno direktne grane, pa je dužina puta dužina grane ako postoji ili $+\infty$ ako ne postoji. Kada znamo (m - 1)-puteve potrebno je da odredimo najkraće m-puteve. Neka je čvor v_m čvor sa rednim brojem m. Najkraći m-put od čvora u do 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 putu od u do v_m i od v_m do v numerisani brojevima manjim ili jednakim od m - 1. Prema induktivnoj hipotezi mi te puteve već znamo. Dakle, da bismo pronašli najkraći m-put od u do v uporedimo najkraći (m - 1)-put od u do v sa zbirom (m - 1)-puteva od u do
v_m i od v_m do v i uzmemo manji. Algoritam unapređuje konstanti faktor u odnosu na algoritam indukcije po broju čvorova ali složenost ostaje $O(|V|^3)$.

Primer:

- U prvom koraku računamo 0-puteve. Put od čvora do samog sebe je inicijalno dužine 0, a za ostale dužina grane ili beskonačno ako grana ne postoji.
- Računamo 1-puteve, odnosno gledamo da li postoji neki put preko čvora 1 koji bi skratio neki trenutno najmanji put. Možemo doći iz 3 u 2 i 3 u 4, a pre nismo mogli, tj. dužina puta je bila beskonačno pa ažuriramo vrednosti.
- Računamo 2-puteve, odnosno gledamo da li postoji neki put preko čvora 2 koji bi skratio neki trenutno najmanji put. Možemo doći iz 1 u 3 što pre nismo mogli pa ažuriramo vrednost. Iz 4 u 3 možemo doći putem dužine 3 što je manje od 0-puta dužine 9 pa ažuriramo vrednost.
- Računamo 3-puteve. Možemo doći iz 2 u 1 i iz 2 u 4 što pre nismo mogli pa ažuriramo vrednosti. Možemo doći iz 4 u 1 što pre nismo mogli pa ažuriramo vrednost.

- Računamo 4-puteve. Iz 1 u 2 možemo doći kraće preko čvora 4, kao i iz čvora 1 u čvor 3 preko čvora 4 i čvora 2 pa ažuriramo vrednosti. Možemo doći iz čvora 3 u čvor 2 kraće preko čvora 4 pa ažuriramo vrednost. Broj čvorova je 4 - kraj algoritma.



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

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

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

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

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

15. Algebarski algoritmi - modularna aritmetika, primeri, primena po izboru

Broj r je **ostatak pri deljenju** broja x brojem y odnosno x mod y = r akko postoji broj q takav da je x = q · y + r, $0 \leq r < y$. Mod može da se koristi i kao oznaka binarne relacije u skupu celih brojeva. Pišemo $a \equiv b \pmod m$ ako m | a - b, odnosno a i b daju isti ostatak pri deljenju sa m, tj. a mod m = b mod m. Na primer, $12 \equiv 2 \pmod 5$ jer 5 | (12 - 2).

Relaciju mod koristimo i u svakodnevnom životu, kao na primer za rad sa vremenom. Na primer, 15 sati nakon 11 časova je 2 sata jer $11 + 15 \equiv 2 \pmod{24}$. Slično važi i za minute, sekunde, dane u nedelji, nedelje i mesece u godini itd. Modularna aritmetika se koristi i za izračunavanje jedinstvenih identifikatora bankovnih računa, u kriptografskim sistemima itd. Brojevi u računaru se takođe predstavljaju po modulu i to po modulu stepena dvojke. Na primer, brojevi tipa unsigned int u C++ su širine 32 bita pa se predstavljaju po modulu 2^{32}.

Relacija mod je **saglasna** sa operacijama sabiranja, oduzimanja i množenja:

- ako $a \equiv b \pmod m$ i $c \equiv d \pmod m$, onda $a \pm c \equiv b \pm d \pmod m$ i $a \cdot c \equiv b \cdot d \pmod m$

Kao posledica važe sledeća dva tvrđenja:

- (a + b) mod m = (a mod m + b mod m) mod m

- (a · b) mod m = (a mod m · b mod m) mod m

Prilikom oduzimanja dva nenegativna broja po modulu m, moguće je dobiti i negativnu vrednost, ali mi želimo da uvek dobijemo nenegativnu vrednost. Neki programski jezici dopuštaju dobijanje i negativnih vrednosti pri moduliranju, a neki ne. Dodavanjem broja m na bilo koji broj po modulu m ne menjamo krajnji rezultat jer je m mod m = 0. Pretpostavimo da imamo brojeve b i a koji su manji od m, tako da je b $\leq$ a. Razlika b - a će onda biti negativna ali kao što smo videli možemo dodati broj m što po modulu m neće promeniti razultat ali hoće dati nenegativan rezultat: (b - a + m) mod m. Kada je b $>$ a važiće isto. U opštem slučaju važi:

- (b - a) mod m = (b mod m - a mod m + m) mod m

Važi i sledeće tvrđenje koje može biti korisno kada želimo da izbegnemo prekoračenja:

- a^n mod m = (a mod m)^n mod m

Primer: odrediti poslednje tri cifre zbira i proizvoda četiri broja.

- Prvo rešenje koje nam pada na pamet je da saberemo i pomnožimo brojeve i da na kraju izračunamo ostatak pri deljenju sa 1000. U slučaju sabiranja to i nije problem, ali kod množenja je moguće da u nekom trenutku dođe do prekoračenja ako su brojevi veliki.
- Znamo da za poslednje tri cifre proizvoda mogu biti relevantne samo poslednje tri cifre množilaca. Dakle, moguće je pre svakog množenja prvo odrediti ostatak pri deljenju sa 1000 množilaca, zatim odrediti proizvod i njegov ostatak pri deljenju sa 1000. Isto će važiti i za sabiranje. Sa $+_m$ možemo označiti sabiranje po modulu m: a $+_m$ b = (a mod m + b mod m) mod m, a sa $\cdot_m$ množenje po modulu m: a $\cdot_m$ b = (a mod m · b mod m) mod m. Rezultat će onda biti ((a $+_m$ b) $+_m$ c) $+_m$ d, odnosno ((a $\cdot_m$ b) $\cdot_m$ c) $\cdot_m$ d.
- Prilikom pisanja konkretnog programskog koda uvek je poželjnije izabrati dobar tip podataka, tj. onaj koji može da "smesti" rezultat jer je operacija računanja ostatka vremenski zahtevna, a u ovoj implementaciji sa javlja često.

16. Algebarski algoritmi - faktorizacija: pojam, algoritam složenosti O($\sqrt{n}$), primena po izboru

Faktorizacija je rastavljanje broja na proste činioce. Na primer, prosti činioci broja 315 su 3, 3, 5 i 7. Problem faktorizacije može da se reši induktivno-rekurzivnim pristupom. Induktivna hipoteza: sve brojeve manje od n umemo da rastavimo na proste činioce. Ako n nije prost onda se može predstaviti kao proizvod d · n_1, gde je d najmanji od svih činilaca broja n (dakle prost broj) i važi d $\leq \sqrt{n}$. Ako bi važilo d > $\sqrt{n}$ onda bi važilo i $n_1 < \sqrt{n}$ pa bi najmanji činilac broja n_1 bio najmanji činilac broja n, suprotno pretpostavci da je d najmanji činilac broja n. Činilac d onda možemo naći prolaskom kroz brojeve od 2 do $\sqrt{n}$, a ostale činioce broja n_1 znamo po induktivnoj hipotezi. Ako je n prost onda je on jedini prost činilac. Dodatno odsecanje možemo izvršiti tako što posebno proverimo broj 2, a onda sve neparne brojeve od 3 do $\sqrt{n}$ jer parni brojevi veći od 2 nisu prosti. U algoritmu nigde eksplicitno ne određujemo proste brojeve, ali redosled kojim tražimo činioce nam garantuje da nikad nećemo uzeti neki složeni broj kao najmanji delilac. Ako imamo složeni broj s = x · y uvek ćemo pre ispitivanja s ispitati x i y jer su manji od s.

Ako su prosti činioci broja n redom $p_1 \leq p_2 \leq \ldots \leq p_{k-1} \leq p_k$, složenost algoritma je O($\max(p_{k-1}, \sqrt{p_k})$). Kada pronađemo sve proste činioce osim dva najveća ostajemo sa brojem n = $p_{k-1} \cdot p_k$. Prost činilac p_{k-1} nalazimo posle O(p_{k-1}) koraka, nakon čega n postaje jednak p_k i provera ide do $\sqrt{p_k}$. Dakle, veća od vrednosti p_{k-1} i $\sqrt{p_k}$ odrediće složenost. U najgorem slučaju, kada je broj prost, p_{k-1} ne postoji pa je složenost jednaka O($\sqrt{p_k}$), odnosno O($\sqrt{n}$).

Primenom ovog algoritma je 2009. godine faktorisan broj od 232 cifre, korišćenjem više stotina računara. Koristi se i za procenu sigurnosti velikog broja kriptografskih algoritama, kao što je RSA algoritam.

17. Algebarski algoritmi - Ojlerova funkcija: pojam, algoritam zasnovan na faktorizaciji, primena po izboru

Ojlerova funkcija broja n - ϕ(n) označava broj prirodnih brojeva manjih ili jednakih n koji su uzajamno prosti sa n. Na primer, ϕ(1) = 1, ϕ(9) = 6 (1, 2, 4, 5, 7, 8), ϕ(17) = 16 jer je 17 prost broj. Kao i faktorizacija, Ojlerova funkcija nalazi primenu u RSA sistemu za šifrovanje, ali se koristi i u teoriji brojeva i algebri.

Problem računanja Ojlerove funkcije nekog broja se može svesti na faktorizaciju broja. Iz definicije Ojlerove funkcije znamo da važi ϕ(1) = 1 i ϕ(p) = p - 1 za prost broj p. Takođe, ako je n = p^k gde je p prost broj onda sa njim nisu uzajamno prosti samo umnošci broja p, tj. brojevi p, 2p, 3p, . . ., p^{k-1} kojih ima ukupno p^{k-1}. Dakle važi: $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$. Važi i $\phi(n \cdot m) = \phi(n) \cdot \phi(m)$, što se može pokazati kineskom teoremom o ostacima. Funkcije za koje važi ovo svojstvo i svojstvo f(1) = 1 (što važi jer ϕ(1) = 1) nazivaju se **multiplikativne funkcije**. Ako broj n u opštem slučaju predstavimo kao $p_1^{k_1} \cdot \ldots \cdot p_r^{k_r}$ gde su $p_1, \ldots, p_r$ različiti prosti činioci broja n na osnovu prethodne dve jednačine važi:

$\phi(n) = \phi(p_1^{k_1})\phi(p_2^{k_2}) \cdot \ldots \cdot \phi(p_r^{k_r}) = p_1^{k_1}(1 - \frac{1}{p_1})p_2^{k_2}(1 - \frac{1}{p_2}) \cdot \ldots \cdot p_r^{k_r}(1 - \frac{1}{p_r}) =$

$p_1^{k_1}p_2^{k_2} \cdot \ldots \cdot p_r^{k_r}(1 - \frac{1}{p_1})(1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_r}) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$

Primećujemo da u krajnjoj formuli figurišu vrednosti prostih činilaca broja n, ali ne i njihovi eksponenti u faktorizaciji. Složenost algoritma odgovara složenosti algoritma faktorizacije, odnosno jednaka je O($\sqrt{n}$).

18. Algebarski algoritmi - prošireni Euklidov algoritam: formulacija problema, izvođenje veza potrebnih za iterativni algoritam, primer, primena po izboru

Osnovni Euklidov algoritam za određivanje najvećeg zajedničkog delioca brojeva a i b: polazeći od r_0 = a i r_1 = b, računamo ostatak $r_2 = r_0 \bmod r_1$, zatim ostatak $r_3 = r_1 \bmod r_2$, itd i na taj način dobijamo opadajući niz ostataka. Za r_i važi $r_{i-1} = q_i r_i + r_{i+1}$. Ovaj niz je konačan jer je opadajući, a sastoji se od prirodnih brojeva. Ako je $r_{k+1} = 0$ i r_k poslednji nenula ostatak, onda važi:

$nzd(r_0, r_1) = nzd(r_1, r_2) = \ldots = nzd(r_{k-1}, r_k) = nzd(r_k, 0) = r_k$

Zaključujemo da je nzd(a, b) upravo jednak poslednjem nenula ostatku tog niza. Uz određeno proširenje, Euklidov algoritam može da se koristi i za rešavanje sledećeg problema:

- Najveći zajednički delilac dva prirodna broja a i b izraziti kao njihovu celobrojnu linearnu kombinaciju, odnosno odrediti cele brojeve x i y tako da važi nzd(a, b) = x · a + y · b.

Algoritam zasnovan na predstavljanju proizvoljnog člana niza ostataka preko a i b

Jednu po jednu vrednost iz niza ostataka možemo predstavljati kao linearnu kombinaciju brojeva a i b, da bismo na kraju došli do r_k, odnosno do nzd(a, b). Znamo da važi $r_0 = a = 1 \cdot a + 0 \cdot b$ i $r_1 = b = 0 \cdot a + 1 \cdot b$. Sada želimo da bilo koji ostatak r_{i+1} predstavimo kao linearnu kombinaciju a i b ako znamo da važi $r_{i-1} = x_{i-1} \cdot a + y_{i-1} \cdot b$ i $r_i = x_i \cdot a + y_i \cdot b$. Na osnovu veze $r_{i+1} = r_{i-1} - q_i r_i$ dobijamo:

r_{i+1} = (x_{i-1} · a + y_{i-1} · b) - q_i(x_i · a + y_i · b) = (x_{i-1} - $q_i x_i$) · a + (y_{i-1} - $q_i y_i$) · b. Dakle važiće veze: x_{i+1} = x_{i-1} - $q_i x_i$ i y_{i+1} = y_{i-1} - $q_i y_i$. Na osnovu ovih veza moguće je implementirati odgovarajući iterativni algoritam. Složenost odgovara složenosti osnovnog Euklidovog algoritma, odnosno jednaka je O($\log(a+b)$).

Primer: nzd(33, 24) predstaviti kao linearnu kombinaciju brojeva 33 i 24.

- r_0 = 33 = 1 · 33 + 0 · 24
- r_1 = 24 = 0 · 33 + 1 · 24
- r_2 = 33 mod 24 = 9 (q_2 = 1) = (1 · 33 + 0 · 24) - 1 · (0 · 33 + 1 · 24) = 1 · 33 - 1 · 24
- r_3 = 24 mod 9 = 6 (q_3 = 2) = (0 · 33 + 1 · 24) - 2 · (1 · 33 1 · 24) = -2 · 33 + 3 · 24
- r_4 = 9 mod 6 = 3 (q_4 = 1) = (1 · 33 - 1 · 24) - 1 · (-2 · 33 + 3 · 24) = 3 · 33 - 4 · 24
- r_5 = 6 mod 3 = 0. Poslednji nenula ostatak je r_4 = 3, pa je nzd(33, 24) = 3 = 3 · 33 - 4 · 24

Prošireni Euklidov algoritam koristi se za rešavanje **linearnih Diofantovih jednačina** koje su oblika: a · x + b · y = c, gde su a, b i c dati celi brojevi, a x i y su celi brojevi koje treba odrediti. Da bi ova jednačina imala rešenje potrebno je da d = nzd(a, b) deli broj c jer on deli levu stranu, pa mora i desnu. Ako je ovo ispunjeno rešenje se dobija tako što se d izrazi kao d = $x' \cdot a + y' \cdot b$, a zatim se obe strane pomnože sa $\frac{c}{d}$ pa se dobija c = $\frac{x'c}{d} \cdot a + \frac{y'c}{d} \cdot b$, pa rešenje polazne jednačine predstavlja par (x, y) = ($\frac{x'c}{d}$, $\frac{y'c}{d}$).

Primer:

- Rešiti jednačinu 4x + 10y = 7. nzd(4, 10) = 2, ali 2 ne deli 7 pa jednačina nema rešenja.
- Rešiti jednačinu 4x + 10y = 8. nzd(4, 10) = 2 i on deli 8 pa jednačina ima rešenja. Korišćenjem proširenog Euklidovog algoritma izražavamo 2 kao linearnu kombinaciju 4 i 10: -2 · 4 + 1 · 10 = 2. Obe strane množimo sa 8/2 = 4, pa dobijamo: -8 · 4 + 4 · 10 = 8, pa je jedno rešenje jednačine (x, y) = (-8, 4).

19. Algebarski algoritmi - modularni multiplikativni inverz, izvođenje rekurzivnog algoritma korišćenjem proširenog Euklidovog algoritma, primer

Multiplikativni inverz broja a po modulu m je broj x za koji važi a · x $\equiv$ 1 (mod m). Kao vrednosti multiplikativnog inverza broja a po modulu m najčešće se razmatraju vrednosti iz skupa {0, 1, 2, . . ., m - 1}. Na primer, multiplikativni inverz 5 po modulu 8 je 5 jer 5 · 5 $\equiv$ 1 (mod 8). Modularni multiplikativni inverz ne mora uvek da postoji, na primer multiplikativni inverz 2 po modulu 8.

Teorema: Ako su brojevi a i m uzajamno prosti pozitivni celi brojevi onda multiplikativni inverz broja a po modulu m postoji i on je jedinstven po modulu m, a inače ne postoji. Naivni pristup pri određivanju bi podrazumevao da prođemo kroz sve brojeve manje od m i nađemo onaj za koji važi a · x $\equiv$ 1 (mod m), što je složenosti O(m). Alternativno, možemo iskoristiti **prošireni Euklidov algoritam**. Preko njega znamo NZD dva broja da predstavimo kao njihovu linearnu kombinaciju: nzd(a, b) = x · a + y · b. Umesto b u jednačinu možemo da uvrstimo m. Pošto pretpostavljamo da su a i m uzajamno prosti važiće nzd(a, m) = 1. Odatle važi x · a + y · m $\equiv$ 1 (mod m). Pošto je y · m deljivo sa m možemo ga ukloniti i na kraju dobijamo x · a $\equiv$ 1 (mod m), odakle sledi da je x multiplikativni inverz broja a po modulu m. Pošto je za x moguće dobiti i negativnu vrednost i vrednost veću od m, kao konačni rezultat možemo vratiti vrednost (x mod m + m) mod m. Na ovaj način smo algoritam traženja multiplikativnog inverza sveli na rekurzivni prošireni Euklidov algoritam. Samim tim i složenost odgovara složenosti proširenog Euklidovog algoritma i iznosi O($\log(a+m)$), odnosno O($\log m$) kada je a < m.

Primer:

- Odrediti MMI 4 po modulu 12. NZD(4, 12) = 4 $\neq$ 1 pa MMI po modulu 12 ne postoji.

- Odrediti MMI 5 po modulu 7. NZD(5, 7) = 1 pa postoji jedinstveni MMI po modulu 7. Primenjujemo prošireni Euklidov algoritam, tj. želimo da nađemo x i y tako da važi 1 = x · 5 + y · 7. Jednakosti koje dobijamo su nzd(7, 5) = nzd(5, 2) = nzd(2, 1). 2 mod 1 je 0 pa je poslednji nenula ostatak 1. Idemo unazad: 1 = 5 - 2 · 2 = 5 - 2 · (7 - 1 · 5) = 3 · 5 - 2 · 7. MMI broja 5 po modulu 7 je 3.

20. Niske - heširanje niski: pojam heš funkcije i kolizije, definicija polinomijalne heš funkcije na dva načina, smanjivanje verovatnoće da dođe do kolizije, primena po izboru

Heširanje niski je konverzija niski u ceo broj iz određenog opsega. Heširanje je korisno u rešavanju različitih problema u radu sa niskama, kao što je ispitivanje da li se jedna niska nalazi u drugoj, pronalaženje najduže niske koja se javlja bar dva puta kao segment druge niske, kompresija niske, određivanje broja palindromskih segmenata niske i slično. Heširanje ima primenu i u verifikaciji lozinki. Lozinke se na serveru čuvaju u heširanom obliku iz bezbednosnih razloga i onda se prilikom logovanja računa heš vrednost unete lozinke i šalje serveru na proveru. Često heširanje može da unapredi algoritme grube sile. Na primer, pri poređenju dve niske umesto poređenja karakter po karakter, možemo izračunati njihove vrednosti pri heširanju i uporediti ih. Konverzija se vrši pomoću **heš funkcije**, a dobijene vrednosti se nazivaju **heš vrednosti**. Ako su dve niske jednake i njihove heš vrednosti moraju biti jednake, međutim obrnuto ne mora da važi jer se može desiti da opseg heš vrednosti nije dovoljno veliki pa dve različite niske mogu imati iste heš vrednosti, što se naziva **kolizija**. Poželjno je da verovatnoća da dođe do kolizije bude što je moguće manja.

Heš funkcija treba da zavisi od multiskupa karaktera koji se javljaju u niski, kao i od njihovog redosleda. Dobar način definisanja heš funkcije niske s dužine n je korišćenjem **polinomijalne heš funkcije**:
h(s) = ($\sum_{i=0}^{n-1}$s[i] · p^{n-i-1}) mod m, gde je s[i] celobrojni kod i-tog karaktera niske, a p i m su unapred odrabani pozitivni brojevi. Da bi se osigurala dobra svojstva heš funkcije parametri p i m trebaju biti pažljivo izabrani. Za parametar p se često uzima prvi prost broj veći od broja karaktera u ulaznoj azbuci (na primer 31 ako se koriste samo slova engleske abecede). Ako bi se uzeo neki broj manji od veličine azbuke onda je on mogući karakter niske i lako se pronalaze niske dužine 2 kod kojih već postoji kolizija. Poželjno je i da parametar m bude neki veliki prost broj jer verovatnoća da dve niske imaju istu heš vrednost je približno jednaka 1/m, ali istovremeno vrednost ne treba biti prevelika. Da heš vrednost ne računamo po nekom modulu parametra m, one bi se računale po modulu broja podržanih vrednosti celobrojnog tipa (2^{32} ili 2^{64}). Parametri p i m se biraju kao prosti brojevi kako bi se smanjio broj kolizija kada niske koje se heširaju ispunjavaju neka matematička svojstva (na primer kada su sve vrednosti umnošci nekog broja). Nekada se za heširanje koristi i naredna heš funkcija: h(s) = ($\sum_{i=0}^{n-1}$s[i] · p^{i}) mod m, gde je najveći stepen broja p uz prvi karakter niske s, a najmanji uz poslednji (obrnuto u odnosu na prvu formulu).

Primer: računanje heš vrednosti niske s, dužine n, koja se sastoji samo od malih slova engleske abecede.

- Postoji 26 mogućih karaktera, pa za njihove konkretne kodove možemo koristiti konverziju a $\rightarrow$ 1, b $\rightarrow$ 2, . . ., z $\rightarrow$ 26. Nije dobro krenuti od a $\rightarrow$ 0 jer bi onda niske a, aa, aaa, . . . imale vrednost heš funkcije 0. Za broj p možemo uzeti prost broj veći od 26, na primer 31. Za broj m možemo uzeti prost broj 10^9 + 9 kako bi što više smanjili broj kolizija. U duhu Hornerove šeme u jednoj petlji možemo ažurirati vrednosti kako prolazimo kroz nisku karakter po karakter, bilo korišćenjem jedne ili druge polinomijalne heš funkcije. Međurezultate možemo računati po modulu m. Jasno je da će složenost biti O(n).

21. Niske - z-algoritam: pojam z-niza, direktno formiranje z-niza, primer izvršavanja z-algoritma, primena po izboru

Z-niz niske s dužine n je niz dužine n koji na poziciji k sadrži dužinu najdužeg segmenta niske s koji počinje na poziciji k i prefiks je niske s. Ako važi z[k] = p onda se segment s[0 . . . p - 1] poklapa sa segmentom s[k . . . k + p - 1] i karakteri s[p] i s[k + p] su različiti ili je niska s dužine k + p. Segment unutar niske koji se preklapa sa nekim prefiksom date niske naziva se **z-kutija**. Vrednost z-niza na poziciji 0 je jednaka dužini niske jer je kompletna niska uvek prefiks same sebe, međutim ta vrednost nije od prevelikog značaja. Z-niz nalazi primenu u rešavanju mnogih problema nad niskama kao što su ispitivanje da li se jedna niska nalazi unutar druge i kompresija niske.

Direktno formiranje z-niza se sastoji u tome da se za svaki indeks iznova traži pozicija najdužeg poklapajućeg prefiksa niske koji počinje na tekućoj poziciji u niski. Za nisku dužine n ovo rešenje bi bilo složenosti $O(n^2)$.
Z-algoritam je algoritam za konstrukciju z-niza u vremenu O(n). Ideja algoritma je da se vrednosti z-niza računaju iterativno, sleva nadesno, na osnovu prethodno izračunatih vrednosti z-niza. U svakom koraku održavamo najdesniji opseg indeksa [l, d] za koji važi da se segment s[l . . . d] poklapa sa nekim prefiksom niske s, odnosno on predstavlja najdesniju z-kutiju. Za računanje vrednosti niza koristimo činjenicu da su segmenti s[0 . . . d - l] i [l . . . d] jednaki. Pri računanju vrednosti z[i] moguća su dva slučaja:

- i > d - tekuća pozicija i je van opsega [l ,d] pa nemamo nikakve informacije o njoj. Vrednost z[i] računamo trivijalnim algoritmom, tj. poredimo karakter po karakter niske s od pozicije i, i od početka niske.
- l < i $\leq$ d - tekuća pozicija i je unutar opsega [l, d] pa možemo iskoristiti neke već izračunate vrednosti. Pošto se segmenti s[0 . . . d - l] i [l . . . d] poklapaju, onda se poklapaju i segmenti s[i . . . d] i [i - l . . . d - l] pa prilikom računanja vrednosti z[i] možemo krenuti od vrednosti z[i - l] koju smo već izračunali. Vrednost z[i - l] u nekim slučajevima može biti prevelika i kada se doda poziciji i može da prevaziđe vrednost indeksa d, što nije dobro jer ne znamo ništa o karakterima niske nakon pozicije d. Dakle, maksimalna dužina poklopljenog dela može biti jednaka broju karaktera od tekuće pozicije i do poslednje pregledane pozicije d, a to je d - i + 1. Početna vrednost za z[i] se onda postavlja na vrednost min{d - i + 1, z[i - l]}, a potom trivijalnim algoritmom proveravamo da li ta vrednost može da se uveća počevši od pozicije i + z[i].
  Ukoliko dođe do poklapanja dela niske počev od pozicije i sa nekim prefiksom date niske i ako je desna granica tog poklopljenog segmenta veća od prethodne vrednosti desne granice, odnosno ako je i + z[i] - 1 > d, ažuriramo granice najdesnije z-kutije [l, d] = [i, i +z[i] - 1].

Primer: Konstruisati z-niz niske ACBACDACBACBACDA.

- Na početku je opseg [l ,d] = [0, 0]. Vrednosti računamo trivijalnim algoritmom i dobijamo z[1] = 0, z[2] = 0, z[3] = 2. Dobili smo novu z-kutiju pa ažuriramo opseg [l, d] = [3, 4].
- Indeks 4 se nalazi u okviru najdesnije z-kutije [3, 4] pa vrednost dobijamo preko već izračunatih vrednosti: z[4] = z[i - l] = z[4 - 3] = z[1] = 0. Vrednosti za indekse 5 i 6 dobijamo trivijalnim algoritmom jer su izvan opsega najdesnije z-kutije i važi z[5] = 0 i z[6] = 5. Ponovo ažuriramo najdesniju kutiju jer za 6 važi i + z[i] - 1 = 6 + 5 - 1 = 10 > d = 4. Nova najdesnija kutija je [6, 10].
- Indeksi 7, 8, 9 i 10 se nalaze u okviru najdesnije z-kutije pa njihove vrednosti računamo preko prethodno izračunatih. Važiće z[7] = z[7 - 6] = z[1] = 0 i z[8] = z[8 - 6] = z[2] = 0. Za indeks 9 dobijamo z[9] = z[9 - 6] = z[3] = 2, ali daljim upoređivanjem karakter po karakter dobijamo z[9] = 7. Važi i + z[i] - 1 = 9 + 7 - 1 = 15 > 10 = d, pa ažuriramo vrednost najdesnije kutije na [9, 15].
- Svi preostali indeksi se nalaze u okviru najdesnije z-kutije pa se i njihove vrednosti računaju na osnovu prethodnih. Važiće z[10] = z[10 - 9] = z[1] = 0, z[11] = z[11 - 9] = z[2] = 0. Za indeks 12 dobijamo z[12] = z[3] = 2 i daljim upoređivanjem ne dobijamo više preklapanja pa je to finalna vrednost. Važi i + z[i] - 1 = 12 + 3 - 1 = 14 < 15 = d pa ne ažuriramo najdesniju kutiju. Na kraju, z[13] = z[4] = 0, z[14] = z[5] = 0 i z[15] = z[6] = 5, ali d - i + 1 = 15 - 15 + 1 = 1, pa je z[15] = 1.

22. Geometrijski algoritmi - skalarni i vektorski proizvod; transformacija polarnih u dekartovske koordinate i obratno, površina trougla: računanje na tri različita načina

Skalarni proizvod dva vektora $\vec{a}$(a_1, . . ., a_n) i $\vec{b}$(b_1, . . ., b_n) je skalar čija se vrednost računa po formuli:

- $\vec{a} \circ \vec{b} = \sum_{i=1}^{n} a_i b_i$

Ako su koordinate vektora celobrojne i skalarni proizvod biće celobrojan. Specijalno, za vektore određene tačkama u ravni važi:

- $\vec{a} \circ \vec{b} = a_x b_x + a_y b_y$
- $\vec{a} \circ \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos\alpha$, gde su prva dva množioca intenziteti vektora, a α ugao koji oni zaklapaju

Iz ove formule moguće je izračunati:

- intenzitet, tj. dužinu vektora $\vec{a}$ kao $|\vec{a}|^2 = \vec{a} \circ \vec{a}$, jer $\cos 0^\circ = 1$
- ugao α između vektora $\vec{a}$ i $\vec{b}$ kao $\alpha = \arccos \frac{\vec{a} \circ \vec{b}}{|\vec{a}| \cdot |\vec{b}|}$

Vektorski proizvod dva vektora $\vec{a}$(a_x, a_y, a_z) i $\vec{b}$(b_x, b_y, b_z) dimenzije 3 je vektor ortogonalan na ravan određenu vektorima $\vec{a}$ i $\vec{b}$, čiji je smer određen pravilom desne ruke, a intenzitet jednak površini paralelgrama koji određuju vektori $\vec{a}$ i $\vec{b}$, odnosno:

- $|\vec{a} \times \vec{b}| = |\vec{a}| \cdot |\vec{b}| \cdot \sin\alpha$, gde je α ugao koji zaklapaju vektori $\vec{a}$ i $\vec{b}$

Ako su koordinate vektora celobrojne i koordinate vektorskog proizvoda biće celobrojne. **Pravilo desne ruke**: ako kažiprst i srednji prst pokazuju redom u smeru vektora $\vec{a}$ i $\vec{b}$, palac će određivati smer njihovog vektorskog proizvoda. Vektorski proizvod se može zapisati matrično:

- $\vec{a} \times \vec{b} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$, gde su $\vec{i}$, $\vec{j}$ i $\vec{k}$ jedinični vektori u smeru x, y i z koordinatne ose. Odnosno, važi:

- $\vec{a} \times \vec{b} = (a_y b_z - a_z b_y)\vec{i} + (a_z b_x - a_x b_z)\vec{j} + (a_x b_y - a_y b_x)\vec{k}$

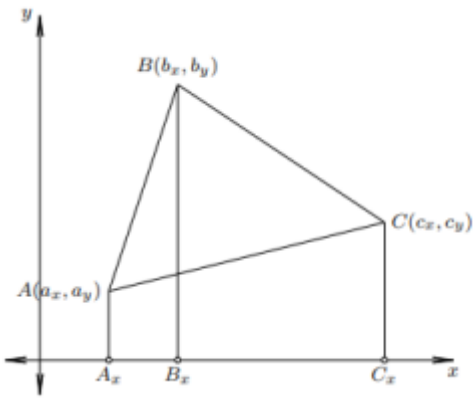
Polarne koordinate tačke T predstavljaju rastojanje r od koordinatnog početka P i ugao ϕ koji vektor $\vec{PT}$ zahvata sa pozitivnim delom x ose. Transformacija polarnih koordinata (r, ϕ) u Dekartove (x, y) i obrnuto:

- $x = r\cos\phi$, $y = r\sin\phi$
- $r = \sqrt{x^2 + y^2}$, $\phi = \arctan\frac{y}{x}$

Računanje površine trougla ABC datog koordinatama svojih temena može se izvršiti na nekoliko načina.

1. **Heronov obrazac**
   - $P = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$, gde su a, b i c dužine stranica trougla, a $s = \frac{a+b+c}{2}$ njegov poluobim. Dužine stranica računamo kao rastojanje između dva temena koja određuju stranicu. U ovom pristupu postoji veliki broj operacija računanja kvadratnog korena i kod računanja dužina stranica i kod samog Heronovog obrasca.

2. **Svođenje na površinu paralelograma**
   - Vektorski proizvod vektora $\vec{AB}$ i $\vec{AC}$ jednak je površini paralelograma koji oni obrazuju. Trougao ABC je polovina tog paralelograma pa mu je površina jednaka polovini intenziteta vektorskog proizvoda. Možemo pretpostaviti da trougao leži u ravni xOy pa za treću koordinatu možemo uzeti 0. Pošto će vektorski proizvod biti normalan na ravan xOy njegove x i y koordinate će biti 0. To znači da u intenzitetu figuriše samo apsolutna vrednost z koordinate. Odnosno, važi da je površina P = $\frac{AB_xAC_y-AB_yAC_x}{2}$ = $\frac{(b_x-a_x)(c_y-a_y)-(b_y-a_y)(c_x-a_x)}{2}$, tj. površina trougla je P = $\frac{b_xc_y-a_xc_y-b_xa_y-bc_x+a_yc_x+a_xb_y}{2}$. Ovo rešenje je efikasnije jer sadrži samo osnovne operacije, a ne korenovanje.
3. **Svođenje na površine trapeza**
   - Ako važi raspored $a_x < b_x < c_x$ i ako su A_x, B_x i C_x projekcije tačaka A, B i C na x-osu, tada je površina trougla ABC jednaka razlici između zbira površina pravouglih trapeza AA_xB_xB i BB_xC_xC i površine pravouglog trapeza AA_xC_xC. Površina trapeza jednaka je proizvodu njegove srednje linije i dužine njegove visine. Na primer, trapez AA_xB_xB ima srednju liniju $\frac{|a_y|+|b_y|}{2}$ i visinu $|b_x - a_x|$, pa je površina jednaka $\frac{|b_x-a_x|(|a_y|+|b_y|)}{2}$. Na isti način se mogu dobiti i druge dve površine. Ispravan rezultat se dobija ako se posmatra i **označena površina** trapeza, koja isključuje računanje apsolutne vrednosti. Označena površina trougla ABC biće jednaka zbiru označenih površina tri pomenuta trapeza, a prava površina apsolutnoj vrednosti označene površine. Dakle, površina je P = $\frac{|(b_x-a_x)(b_y+a_y)+(c_x-b_x)(c_y+b_y)+(a_x-c_x)(a_y+c_y)|}{2}$ = $\frac{|b_xa_y-a_xb_y+c_xb_y-b_xc_y+a_xc_y-c_xa_y|}{2}$.

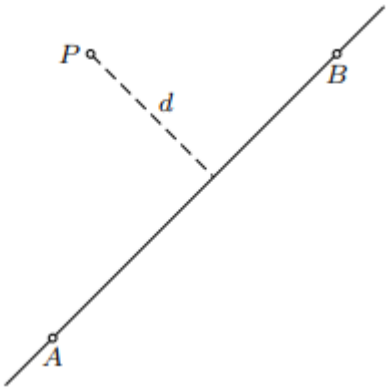


U poslednje dve metode smo dobili istu formulu koja koristi **pravilo pertle**. Ako koordinate zapišemo u narednom obliku:

a_x a_y
b_x b_y
c_x c_y
a_x a_y

formula se gradi tako što se sa jednim znakom uzimaju proizvodi odozgo-naniže (a_xb_y, b_xc_y i c_xa_y), a sa suprotnim znakom proizvodi odozdo-naviše (a_xc_y, c_xb_y i b_xa_y), što podseća na cik-cak vezivanje pertli.

23. Geometrijski algoritmi - računanje rastojanja tačke od prave, ispitivanje kolinearnosti tačaka, utvrđivanje da li tačka pripada trouglu

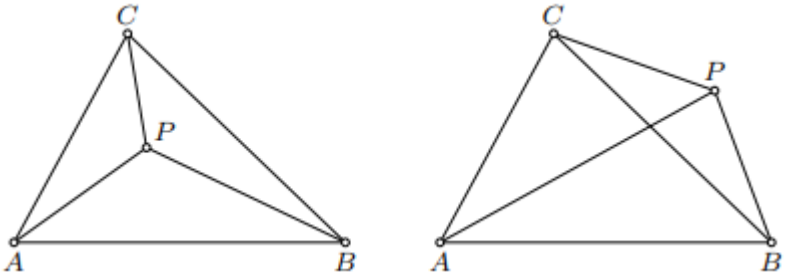


Rastojanje tačke P od prave određene tačkama A i B možemo računati na dva načina:

- Površina trougla PAB je jednaka P = $\frac{|\vec{AB}|\cdot d}{2}$ = $\frac{|\vec{PA}\times\vec{PB}|}{2}$, pa je rastojanje d = $\frac{|\vec{PA}\times\vec{PB}|}{|\vec{AB}|}$
- Odredimo implicitnu jednačinu prave kroz tačke A i B (ax + by + c = 0), a zatim rastojanje tačke P(x_0, y_0) od te prave računamo kao d = $\frac{ax_0+by_0+c}{\sqrt{a^2+b^2}}$

Kolinearnost tačaka A(a_x, a_y), B(b_x, b_y) i C(c_x, c_y) se može odrediti na dva načina:

- **Računanje nagiba pravih** - tačke su kolinearne ako je nagib prave određene tačkama A i B jednak nagibu prave određene tačkama A i C. Nagib prave jednak je količniku razlike y i x koordinata tačaka kojima je prava određena. Dakle, tačke su kolinearne ako važi $\frac{b_y-a_y}{b_x-a_x}$ = $\frac{c_y-a_y}{c_x-a_x}$, odnosno ako $(b_y-a_y)(c_x-a_x)=(c_y-a_y)(b_x-a_x)$.
- **Računanje površine trougla** - tačke su kolinearne ako je površina trougla ABC jednaka 0. Dakle, tačke su kolinearne ako važi P = $\frac{|b_xa_y-a_xb_y+c_xb_y-b_xc_y+a_xc_y-c_xa_y|}{2}$ = 0.



Utvrđivanje da li tačka pripada trouglu se može izvršiti na dva načina:

- **Svođenje na površine trouglova** - ako je tačka P u unutrašnjosti trougla ABC onda je zbir površina trouglova PAB, PBC i PCA jednak površini trougla ABC
- **Računanje orijentacije trojki tačaka** - tačka P se nalazi u unutrašnjosti trougla ABC akko su trouglovi ABP, BCP i CAP iste orijentacije. **(objašnjenje orijentacije u sledećem pitanju)**

24. Geometrijski algoritmi - utvrđivanje orijentacije trojke tačaka u ravni i njene primene

Uređena trojka tačaka u ravni može biti kolinearna ili imati pozitivnu (u smeru suprotnom od kretanja kazaljke sata) ili negativnu orijentaciju (u smeru kretanja kazaljke sata). Ako trojka A, B, C ima pozitivnu (negativnu) orijentaciju, onda i trojka B, C, A ima pozitivnu (negativnu) orijentaciju, a trojka B, A, C ima negativnu (pozitivnu) orijentaciju.

Trougao ABC ima pozitivnu orijentaciju ako vektorski proizvod vektora $\vec{AB}$ i $\vec{AC}$ ima pozitivnu vrednost. Kao i u slučaju računanja površine trougla, za treću koordinatu tačaka uzimamo 0, pa je vektorski proizvod normalan na ravan xOy, pa je za vrednost vektorskog proizvoda bitna samo z koordinata. Dakle, ako je $(b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$ > 0 orijentacija je pozitivna, ako je < 0 onda je negativna, a inače su tačke kolinearne.

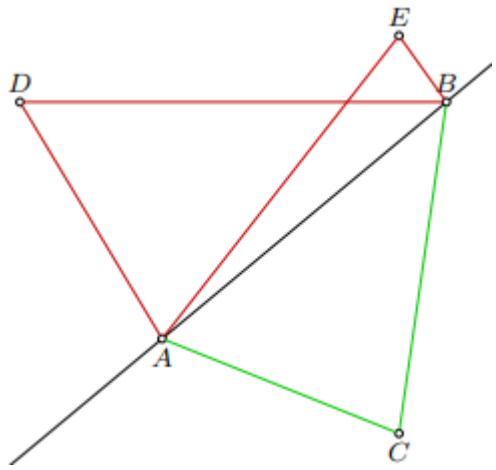
Primene orijentacije:

Utvrditi da li tačka P pripada trouglu ABC (prethodno pitanje)

- Tačka P se nalazi u unutrašnjosti trougla ABC akko su trouglovi ABP, BCP i CAP iste orijentacije.

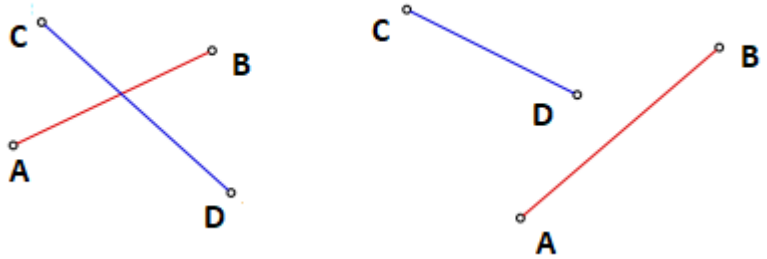
Za tačke C i D utvrditi da li su sa iste strane prave p određene tačkama A i B

- Tačke C i D biće sa iste strane prave p akko su trouglovi ABC i ABD iste orijentacije.

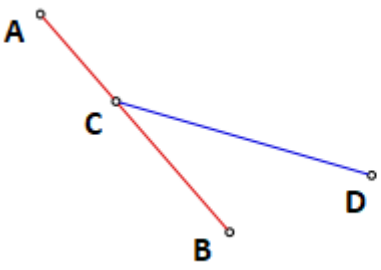


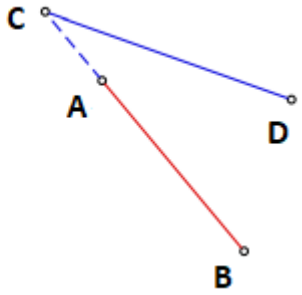
Odrediti da li se dve duži AB i CD seku. Duži AB i CD se seku akko važi jedan od sledećih uslova:

- nikoje tri tačke nisu kolinearne i važi da su tačke C i D sa različitih strana prave AB i tačke A i B su sa različitih strana prave CD. (levo važi, desno ne važi)

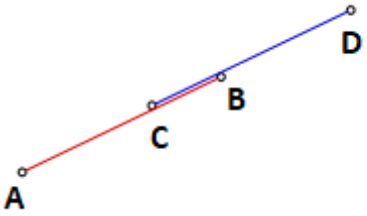


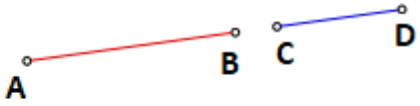
- tačno tri tračke su kolinearne, npr. A, B i C, i važi da je tačka C između tačaka A i B. (levo važi, desno ne važi)





- Sve tačke su kolinearne, a seku se i projekcije duži AB i CD na x-osu i projekcije duži na y-osu. (levo važi, desno ne važi)





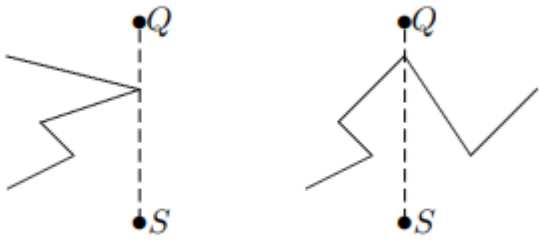
25. Geometrijski algoritmi - utvrđivanje da li tačka pripada proizvoljnom prostom mnogouglu, analiza složenosti, primer

Prvi intuitivni pristup jeste pokušati "izaći napolje" polazeći od date tačke Q. Možemo posmatrati proizvoljnu polupravu iz te tačke i brojati preseke sa stranicama mnogougla. Lako se zaključi da je tačka Q u mnogouglu akko je broj preseka poluprave iz Q sa stranicama mnogougla neparan. Ako je broj preseka paran, tačka se nalazi van mnogougla. Ovaj pristup je lak kada radimo sa slikom, ali ne i kada nam je mnogougao zadat samo preko koordinata svojih temena. Algoritam možemo uprostiti ako umesto proizvoljne poluprave izaberemo polupravu paralelnu jednoj od osa - na primer polupravu paralelnu y-osi usmerenu nadole. Presek te poluprave sa stranicom mnogougla P_iP_{i+1} će postojati ako se x koordinata tačke Q nalazi između x koordinata P_i i P_{i+1} i ako je y koordinata presečne tačke manja od y koordinate tačke Q (jer smo uzeli polupravu nadole). Poluprava će imati jednačinu $x = x_Q$, a prava P_iP_{i+1} jednačinu $y - y_{P_i} = \frac{y_{P_{i+1}} - y_{P_i}}{x_{P_{i+1}} - x_{P_i}}(x - x_{P_i})$. Kada se u jednačinu ubaci x_Q umesto x, dobija se vrednost y koordinate presečne tačke. Za nju mora da važi $y \leq y_Q$ pa se sređivanjem izraza dobija uslov $(y_{P_i} - y_Q)(x_{P_{i+1}} - x_{P_i}) - (y_{P_{i+1}} - y_{P_i})(x_{P_i} - x_Q) \leq 0$.

Obično postoje i neki slučajevi koje treba posebno razmotriti. Pretpostavimo da sada preseke sa stranicama mnogougla ne brojimo preko poluprave nego preko neke duži QS, gde je S tačka koja se nalazi van mnogougla. Prvi specijalni slučaj je kada se duž QS delom preklapa sa nekom stranicom mnogougla. Jasno je da ovo preklapanje ne ubrajamo u preseke. Drugi specijalni slučaj je kada duž QS sadrži neko teme P_i mnogougla. Postoje dva načina na koji se može rešiti ovaj slučaj.

- Ako su dva temena susedna temenu P_i sa iste strane prave kojoj pripada duž QS onda se presek ne računa, a ako su sa različitih strana presek se broji. Na levoj slici su sa iste strane i ne računamo presek, dok na drugoj računamo jer su sa različitih strana.
- Za svaku stranicu računamo presek samo sa levim temenom, a sa desnim ne. Levo i desno teme se određuje na osnovu vrednosti x koordinata temena. Na levoj slici nećemo brojati presek jer je za obe stranice

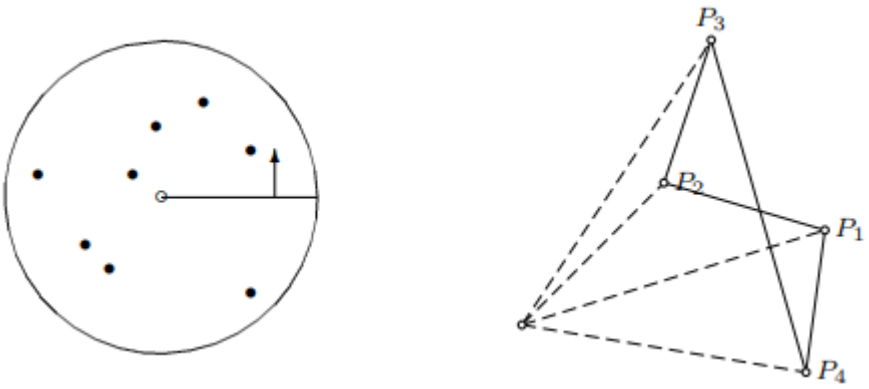
koje sadrže teme to teme desno teme. Na desnoj slici jednoj stranici je to teme levo, a drugoj desno, pa ga računamo samo jednom.



U algoritmu prolazimo kroz sve stranice mnogougla i proveravamo da li imamo presek ili ne. Ove provere su konstantne složenosti pa je ukupna složenost O(n) gde je n broj stranica mnogougla. **(za primer uzeti bilo koji mnogougao, pravu QS sa nekim specijalnim slučajem i primeniti algoritam)**

26. Geometrijski algoritmi - konstrukcija prostog mnogougla, analiza složenosti, primer

Postoji više metoda za konstrukciju prostog mnogougla od datih n tačaka. **Geometrijski pristup** podrazumeva nalaženje kruga koji bi sadržao sve tačke. Za nalaženje tog kruga dovoljno je O(n) operacija koje podrazumevaju izračunavanje najvećeg među rastojanjima proizvoljne tačke ravni do svih n tačaka. Površinu kruga onda "pregledamo" rotirajućom polupravom kojoj je početak centar kruga. Pretpostavimo da poluprava u svakom trenutku sadrži najviše jednu tačku, onda očekujemo da ćemo spajanjem tačaka onim redosledom kojim poluprava nailazi na njih dobiti prost mnogougao. Ovaj pristup je na prvi pogled dobar ali prilikom nekih loših izbora centra kruga možemo dobiti mnogougao koji nije prost. Bolji način odabira centra kruga može se izvršiti tako što fiksiramo proizvoljne tri tačke i za centar kruga izaberemo neku tačku unutar trougla koji one formiraju (na primer težište). Sada tačke sortiramo prema položaju u krugu, tačnije sortiramo uglove između x-ose i polupravih od centra kruga ka tačkama. Ako neke tačke imaju isti ugao one se sortiraju rastuće prema rastojanju od centra kruga. Tačke na kraju povezujemo dobijenim uređenjem. Ugao ϕ računamo kao $\arctan k$ gde je k koeficijent prave određene centrom kruga i tačkom koja se posmatra. Ako prave imaju isti ugao poredimo rastojanja od centra i pritom možemo porediti kvadrate rastojanja kako ne bismo imali operacije korenovanja. Osnovna složenost algoritma potiče od sortiranja pa je jasno da iznosi O($n \log n$).



- Na levoj slici je centar izabran kao težište neke tri tačke iz skupa. Tačke su sortirane prema uglovima koje zaklapaju prave od centra ka njima sa x-osom. Ako se tačke povežu videćemo da smo dobili prost mnogougao.
- Na desnoj slici je centar izabran proizvoljno, i to jako loše. Na kraju je dobijen mnogougao koji nije prost.

Umesto težista proizvoljnog trougla za centar kruga se može uzeti i **ekstremna tačka**, odnosno tačka sa najvećom x-koordinatom i najmanjom y-koordinatom. Ovu tačku povežemo sa svim ostalim tačkama i onda ih sortiramo prema uglu koji te prave zaklapaju sa x-osom. Ako dve ili više tačaka zaklapaju isti ugao sa x-osom, one se sortiraju prema rastojanju od centra kruga: ako prvih nekoliko tačaka zaklapa isti ugao sortiramo ih rastuće, ako poslednjih nekoliko tačaka zaklapa isti ugao sortiramo ih opadajuće, dok je za ostale tačke svejedno. Umesto uglova moguće je koristiti orijentaciju tačaka. Tačka P_i zaklapa manji ugao od tačke P_j ako je orijentacija trojke tačaka z, P_i, P_j pozitivna, gde je z centar kruga.

27. Geometrijski algoritmi - konstrukcija konveksnog omotača – algoritam po izboru, primer

(ispod je algoritam uvijanja poklona, Grejemov algoritam se nalazi u skripti za osnovni nivo, pitanje broj 22)

Konveksni omotač konačnog skupa tačaka je najmanji konveksni mnogougao koji sadrži sve tačke tog skupa.

Algoritam uvijanja poklona: Krećemo od ekstremne tačke, npr. tačke sa najmanjom x-koordinatom i najmanjom y-koordinatom. Ekstremna tačka uvek pripada konveksnom omotaču. U svakom koraku dodajemo novo teme, tj. novu stranicu konveksnog omotača. Jedno teme nove stranice biće poslednja tačka do sada određenog dela konveksnog omotača. Drugo teme nove stranice tražimo među preostalim tačkama i biramo onu koja daje stranicu koja sa prethodnom stranicom gradi što veći ugao. U polaznom slučaju nema prethodne duži, pa tražimo tačku koja gradi najveći ugao sa negativnim delom y-ose. Nije potrebno eksplicitno računati uglove. Ako je poslednja dodata tačka A, onda tačka A_i gradi veći ugao od tačke A_j akko je orijentacija trojke A, A_i, A_j negativna. Ako postoji više tačaka sa istim uglom, onda se uzima ona koja je najdalja od poslednjeg dodatog temena omotača. Postupak se završava kada dođemo do situacije da je tačka koja daje najveći ugao sa prethodnom tačkom jednaka početnoj tački. Algoritam je poznat i pod nazivom **Džarvisov marš**. Algoritam je posledica primene induktivne hipoteze po k: za dati skup od n tačaka u ravni, umemo da pronađemo konveksni put dužine k < n koji je deo konačnog konveksnog omotača datog skupa tačaka. Naglasak je na proširivanju puta, a ne omotača. Umesto pronalaženja konveksnih omotača podskupova, mi pronalazimo delove konačnog konveksnog omotača.

Primer: **pogledati** Predavanje 13 **od 2:20:05 do 2:27:35**