Konstrukcija i analiza algoritama

Osnovni nivo

1. Prefiksno drvo - implementacija, složenost osnovnih operacija, poređenje sa balansiranim binarnim stablima i heš tabelama

Primer implementacije prefiksnog drveta za formiranje i pretragu rečnika:

- Čvorovi prefiksnog drveta mogu imati različit broj dece, ali je maksimalni broj ograničen veličinom azbuke koja se koristi za kodiranje ključeva. S obzirom da ne mora svaki čvor imati sve moguće grane efikasno rešenje bi bilo da se koristi mapa. Dakle, u svakom čvoru čuvamo neuređenu (heš) mapu grana koje kreću iz tog čvora obeleženih karakterima (slovima azbuke). Prilikom implementacije traženih operacija prolazimo kroz čvorove, tj. kroz njihove mape.

Vremenska složenost:

- U slučaju kada je azbuka veličine K, a dužina reči M onda je složenost umetanja, brisanja ili pretraživanja te reči u najgorem slučaju O(MK) jer je soženost najgoreg slučaja pretrage neuređene mape O(K) a prilikom obrade ključa dužine M vrši se M takvih pretraga. Amortizovana složenost pretrage neuređene mape je O(1) pa je amortizovana složenost operacija O(M). Složenost najgoreg slučaja je takođe O(M) ako se radi sa niskama karaktera i ako se naurređena mapa implementira pomoću niza od m elemenata.

Prostorna složenost:

- Prednost prefiksnog drveta je to što složenost zavisi od dužine zapisa ključa, a ne od broja elemenata koji se čuvaju u drvetu. Mana je potreba za čuvanjem pokazivača uz svaki čvor. Prostorna složenost drveta sa N ključeva u najgorem slučaju iznosi O(MNK), gde je M maksimlna dužina ključa, jer je maksimalni broj čvorova O(MN) (kada nema nikakvog preklapanja karaktera među ključevima) a prostorna složenost svakog čvora je O(K) zbog čuvanja mape. U stvarnom životu je očekivana složenost manja jer se preklapanja dešavaju dosta rano (na primer, u slučaju engleske abecede već nekon 26 ključeva). Smanjenje memorijske složenosti se može postići i smanjivanjem veličine azbuke, po cenu povećanja dužine ključa.

Poređenje:

- Ako bi se umesto prefiksnog drveta koristilo balansirano uređeno binarno drvo koje bi čuvalo ključeve u čvorovima vremenska složenost osnovnih operacija bi u najgorem slučaju bila O(M$\log N$) gde je N ukupan broj ključeva u drvetu, a M maksimalna dužina ključa. Prostorna složenost bi bila O(MN).
- Ako bi se umesto prefiksnog drveta koristila heš tabela za svaku operaciju bi morala da se računa heš vrednost za šta je potrebno vreme O(M). U zavisnosti od broja kolizija u najgorem slučaju bi se vršilo O(N) poređenja heš vrednosti, a amortizovano O(1). Na kraju bi ključevi biti eksplicitno poređeni za šta je potrebno vreme O(M) tako da bi ukupna složenost bila O(M + N), a amortizovana složenost O(M). Prostorna složenost b bila O(MN) jer ključ mora biti zapisan u svakom slogu tabele.

2. Disjunktni skupovi (union-find) - opis efikasne strukture podataka i operacija nad njom, primeri

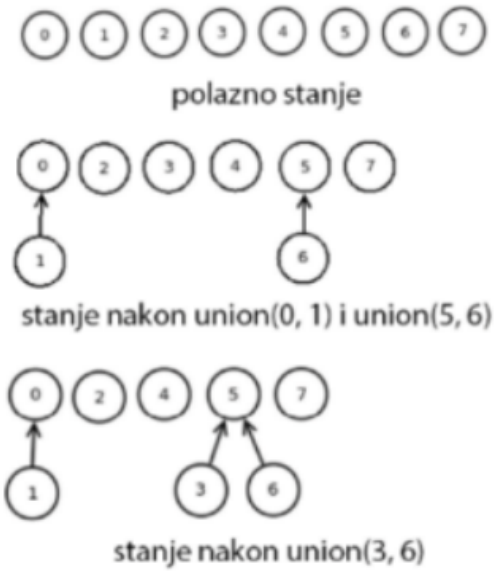
Nekada je u programu potrebno održavati nekoliko disjunktnih podskupova nekog skupa pri čemu je potrebno efikasno izvršiti dva tipa operacija:

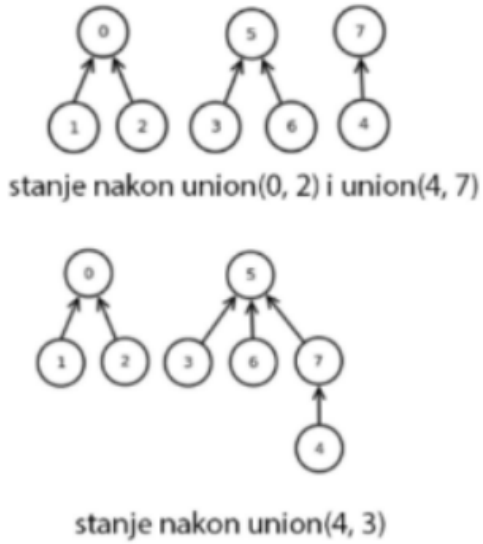
- **find** - pronaći kom podskupu pripada dati element.
- **union** - spojiti dva data podskupa u jedan veći podskup. Argumenti ove operaciju ne moraju biti dva podskupa, već dva elementa koja pripadaju tim podskupovima.

Ovakva struktura podataka se naziva **union-find** ili **DSU** (disjoint set union). Jedan primer korišćenja ove strukture je rad sa relacijama ekvivalencije. Provera da li su dva elementa u relaciji ekvivalencije se zasniva na proveri da li pripadaju istoj klasi, a uspostavljanje relacije između dva elementa dovodi do spajanja njihovih klasa ekvivalencije.

- **Naivna implementacija**: Ova implementacija podrazumeva da se održava preslikavanje svakog elementa u oznaku podskupa kojem pripada. Preslikavanje se može realizovati pomoću običnog niza ako elemente numerišemo brojevima od 0 do n - 1 za skup od n elemenata. Ako elementi nisu numerisani brojevima može se koristiti i mapa. Operacija find je trivijalna jer se oznaka podskupa samo pročita u vremenu O(1). Operacija union je složenija jer moramo proći kroz ceo niz da bismo promenili potrebne oznake elementima i zahteva vreme O(n). Dakle ako bismo imali m operacija koje treba izvršiti složenost bi bila O(mn).
- **Efikasna implementacija**: Ključna ideja ove implementacije je da podskupove čuvamo u obliku drveta tako da svaki element slikamo u njegovog roditelja u drvetu. Korene drveta ćemo slikati same u sebe i smatrati ih oznakama podskupova predstavljenih tim drvetom. U ovom drvetu pokazivači su usmereni od dece ka roditeljima, suprotno od klasičnih drveta. Operaciju find implementiramo tako što od datog elementa prolazimo kroz niz roditelja sve dok ne stignemo do korena. Operaciju union realizujemo tako što koren jednog podskupa usmerimo ka korenu drugog. Ukoliko bi ova drveta ostala izbalansirana u svakom koraku, složenost obe operacije je O($logn$) pa je za m upita složenost O(m$\log n$) što je bolje u odnosu na prvi slučaj. To se postiže tako što se tokom spajanja dva podskupa koren plićeg podskupa preusmerava ka korenu dubljeg podskupa i tada će se visina povećati samo ako su oni iste visine.

Primer: operacije union na skupu od 8 elemenata.





3. Segmentna drveta - implementacija osnovnih operacija, analiza složenosti operacija

Računanje zbira segmenta

- **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. U svakom koraku se dužina segmenta polovi, a inicijalno je manja ili jednaka n pa je složenost operacije O($\log n$).
- **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. Složenost ovog pristupa je takođe O($\log n$). Na prvom nivou se posećuje 1 čvor. Ako se posećuje najviše 2 čvora u narednom nivou se posećuje najviše četiri čvora jer svaki čvor može da proizvede najviše 2 rekurzivna poziva. Ako se posećuju 3 ili 4 čvora oni mogu biti susedni, ali i ne moraju. Čvorovi na jednom nivou drveta sadrže sume disjunktnih segmenata pa jedini čvorovi od kojih se mogu pokrenuti rekurzivni pozivi su oni koji sadrže granice segmenta, a ostali se ili potpuno sadrže u traženom segmentu ili su disjunktni. Dakle i u ovom slučaju postoje najviše 2 čvora od kojih se može pokrenuti rekurzija što opet daje 4 čvora na sledećem nivou. Složenost je najviše O($4\log n$) što je zapravo O($\log n$).

Ažuriranje vrednosti elementa

- **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. Složenost ove operacije je O($\log n$) jer se k polovi u svakom koraku, a najveća moguća vrednost mu je 2n - 1.
- **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). Složenost ove implementacije se može opisati jednačinom T(n) = T($\frac{n}{2}$) + O(1), T(1) = O(1) jer se dužina intervala u svakom pozivu smanjuje dva puta, a maksimalna dužina mu je n. Iz master teoreme dobijamo da je rešenje jednačine O($\log n$).

4. Grafovi - ispitivanje da li usmereni graf sadrži usmereni ciklus svođenjem na DFS numeraciju čvorova

Lema: Neka je G = (V, E) usmereni graf i neka je T njegovo DFS drvo. Graf G sadrži ciklus akko graf G sadrži povratnu granu u odnosu na DFS drvo T.

Dokaz:

- $\Leftarrow$: Pretpostavimo da graf sadrži povratnu granu (u, v). Ona zajedno sa granama DFS drveta na putu od v do u čini ciklus. □
- $\Rightarrow$: Pretpostavimo da u grafu postoji ciklus (v_1, v_2), (v_2, v_3), . . ., (v_{k-1}, v_k), (v_k, v_1) čija nijedna grana nije povratna u odnosu na drvo T. Ako je k = 1, tj. ciklus je petlja onda je grana (v_1, v_1) povratna, suprotno pretpostavci. Ako je k > 1 prepostavimo da ni jedna od grana (v_1, v_2), (v_2, v_3), . . ., (v_{k-1}, v_k) nije povratna. Onda će važiti v_1.Post > v_2.Post > . . . > v_k.Post. Iz v_k.Post < v_1.Post sledi da je grana (v_k, v_1) povratna, suprotno pretpostavci. □

Iz ove leme sledi da se algoritam za proveru da li graf sadrži ciklus može svesti na DFS pretragu, odnosno na određivanje odlazne DFS numeracije čvorova i proveru postojanja povratne grane. Složenost ovog algoritma je O(|V|, + |E|).

5. Grafovi - topološko sortiranje - pojam, Kanov algoritam, složenost algoritma, 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.

Kanov algoritam se bazira na induktivnoj hipotezi da umemo da topološki sortiramo čvorove svih usmerenih acikličnih grafova sa manje od n čvorova. Bazni slučaj je kada graf sadrži jedan čvor što se rešava trivijalno. Dakle, iz početnog grafa treba ukloniti jedan čvor i po hipotezi onda znamo da sortiramo n - 1 čvorova, pa na kraju proširirmo numeraciju na polazni graf. Potrebno je ukloniti čvor za koji bismo najlakše proširili induktivnu hipotezu. To je upravo čvor koji ne zavisi od drugih čvorova, odnosno **čvor ulaznog stepena nula**. Postavlja se pitanje da li graf ima takav čvor.

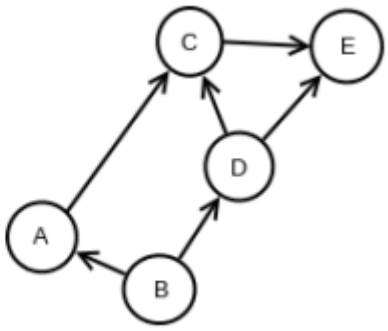
Lema: Usmereni aciklički graf uvek ima čvor ulaznog stepena nula.

- **Dokaz**: Pretpostavimo da u usmerenom acikličkom grafu svi čvorovi imaju ulazne stepene veće od nule. Ako je to slučaj onda počevši od nekog čvora možemo ići unazad kroz grane , a pošto je broj čvorova konačan jednom moramo doći u neki čvor po drugi put. To znači da u grafu postoji ciklus, suprotno pretpostavci. □

Dakle, pronalazimo čvor ulaznog stepena nula, numerišemo ga sa 1 i njega izbacujemo iz grafa. Preostalih n - 1 čvorova numerišemo brojevima od 2 do n (po induktivnoj hipotezi znamo da ih numerišemo brojevima od 1 do n - 1, pa taj broj samo uvećamo za jedan). Ulazne stepene svih čvorova možemo čuvati u posebnom nizu i računati ga jednostavim prolaskom kroz listu povezanosti u linearnoj složenosti po broju grana. Kada naiđemo na granu (u, w) jasno je da treba povećati ulazni stepen čvora w za jedan. Takođe, sve trenutne čvorove stepena nula trebamo posebno čuvati jer želimo da ih obrađujemo po nekom redosledu. Možemo ih čuvati u redu ili steku. Po lemi u prvom koraku algoritma postoji bar jedan čvor stepena nula. Njega skidamo sa reda i označavamo sa 1, a zatim prolazimo kroz sve grane koje polaze od njega i krajnjim čvorovima smanjujemo ulazni stepen za jedan, odnosno uklanjamo tu granu iz grafa (nije potrebno bukvalno menjati listu povezanosti, već samo ažurirati niz sa ulaznim stepenima čvorova). Ako neki od tih čvorova sada ima ulazni stepen 0 on se dodaje u red. Algoritam se završava kada red sa čvorovima stepena nula postane prazan i tada će svi čvorovi biti numerisani.

Ako je graf zadat listama povezanosti, onda je vremenska složenost određivanja niza ulaznih stepena O(|V| + |E|). Prilikom uklanjanja čvorova jasno je da se ukupno svaka grana razmatra jednom pa je ukupan broj promena vrednosti ovog niza jednak O(|E|). Dakle, ukupna složenost Kanovog algoritma je O(|V| + |E|).

Primer



- Prvo određujemo ulazne stepene čvorova:

čvor	A	B	C	D	E
ulazni stepen	1	0	2	1	2

- Čvor B ima ulazni stepen 0 pa ga uklanjamo i označavamo sa 1. Ulazni čvorovi za A i D će se smanjiti za 1, jer "brišemo" grane iz čvora B.
- Čvorovi A i D sada imaju ulazni stepen 0 pa možemo ukloniti bilo koji od njih. Uklanjamo čvor A i označavamo ga sa 2. Smanjujemo ulazni stepen čvora C za 1.
- Čvor D ima ulazni stepen 0 pa ga uklanjamo i označavamo sa 3. Smanjujemo ulazne stepene čvorova C i E za 1.
- Čvor C ima ulazni stepen 0 pa ga uklanjamo i označavamo sa 4. Smanjujemo ulazni stepen čvora E za 1.
- Čvor E ima ulazni stepen 0 pa ga uklanjamo i označavamo sa 5.
- Red je prazan, dakle algoritam se završava i svi čvorovi su označeni. Dobijeni poredak je B, A, D, C, E. Da smo prvo izbacili čvor D dobili bi poredak B, D, A, C, E.

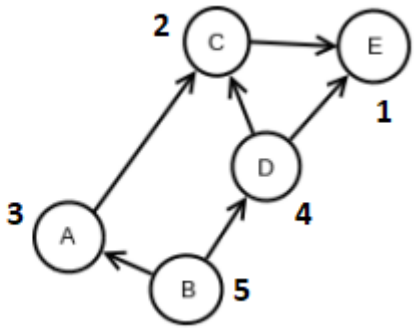
6. Grafovi - topološko sortiranje - pojam, algoritam zasnovan na pretrazi u dubinu, složenost algoritma, primer

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. To jednostavno možemo uraditi tako što na kraju DFS pretrage samo obrnemo redosled kojim su označeni.

Pošto se algoritam svodi na DFS pretragu jasno je da će vremenska složenost biti O(|E| + |V|).

Primer

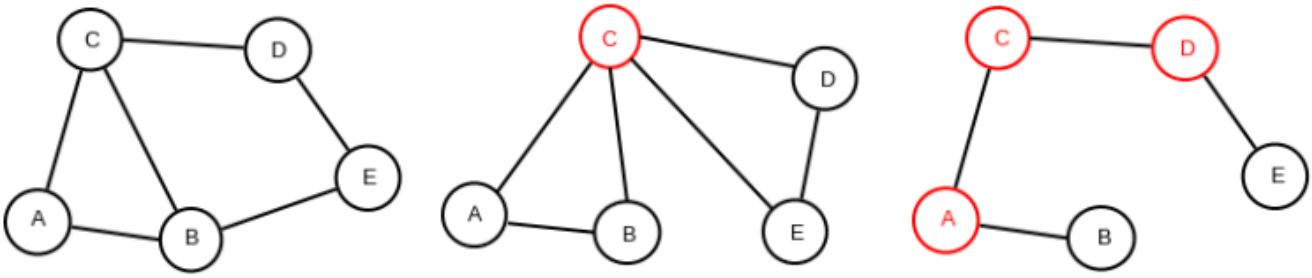


- DFS pretragu možemo pokrenuti iz čvora B. Odlazna numeracija će dati redosled E, C, A, D, B. Kada obrnemo redosled dobijamo poredak B, D, A, C, E što odgovara traženom topološkom uređenju.

7. Grafovi - artikulacione tačke - pojam, karakterizacija, određivanje svih artikulacionih tačaka u grafu, primer

Artikulaciona tačka je čvor grafa čijim se uklanjanjem (a sa njim i grana koje su mu susedne) broj komponenti povezanosti grafa povećava. Dakle, povezan graf uklanjanjem artikulacione tačke postaje nepovezan. Graf može da ne sadrži artikulacione tačke, a može ih imati i više od jedne.

Primer: graf bez, sa jednom i sa tri artikulacione tačke.



Direktan način da se u neusmerenom povezanom grafu pronađu sve artikulacione tačke bi podrazumevao da izbacujemo jedan po jedan čvor i za svaki proveravamo da li je graf povezan bez njega, npr. korišćenjem DFS pretrage. Složenost tog pristupa bi bila O(|V| · (|V| + |E|)).

Tardžanov algoritam za traženje artikulacionih tačaka

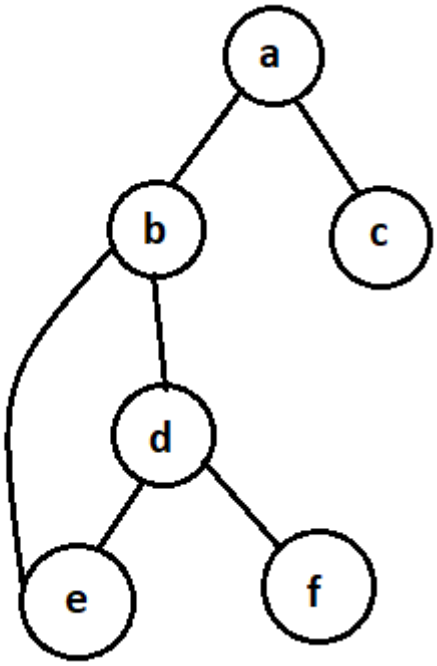
(prvo naučiti algoritam za mostove u narednom pitanju!)

Čvor će biti artikulaciona tačka ako važi neki od sledećih uslova:

- u je koren DFS drveta i ima bar dva deteta - pošto kod neusmerenih grafova ne postoje poprečne grane izbacivanje korena DFS drveta bi jasno dovelo da povećanja komponenti povezanosti. Ovaj slučaj možemo ispitati tako što za svaki čvor prilikom DFS pretrage ispitujemo da li ima roditelja i koliko dece ima. Ako čvor nema roditelja i ako ima bar dva deteta onda se radi o korenu koji ima bar dva deteta tj. o artikulacionoj tački.
- u nije koren DFS drveta i ima dete v u DFS drvetu takvo da nijedan čvor u poddrvetu čvora v nije povezan sa nekim pretkom čvora u u DFS drvetu - ako bismo izbacili u onda ni iz jednog čvora njegovog poddrveta ne možemo doći do nekog pretka u, pa je jasno da će njegovo poddrvo ostati "izolovano" od ostatka grafa. Ovaj slučaj možemo ispitati pomoću **low link** vrednosti, slično kao kod algoritma za mostove. Čvor u (koji nije koren DFS drveta) je artikulaciona tačka ako za neko njegovo dete v važi L(v) $\geq$ u.Pre.

Složenost algoritma je ponovo O(|V| + |E|) kao kod mostova, jer se zasniva na DFS pretrazi.

Primer: sledeći graf je zadat listom povezanosti gde su čvorovi u listama uređeni leksikografski.

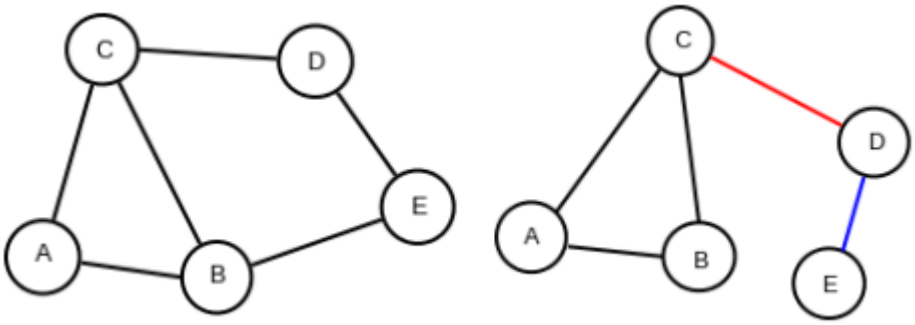


- Pokrećemo DFS pretragu iz čvora a, pa čvor a nema roditelja. Low link i dolazna numeracija su mu 1. On ima 2 sina pa je samim tim artikulaciona tačka. Sledeći sused je čvor b.
- Pokrećemo DFS iz čvora b, on ima roditelja (svi naredni čvorovi imaju roditelja). Low link i dolazna numeracija su mu 2. Njegov sused a je roditelj. Idemo u sused d.
- Pokrećemo DFS iz čvora d, low link i dolazna numeracija su mu 3. Njegov sused b je roditelj. Idemo u sused e.
- Pokrećemo DFS iz čvora e, low link i dolazna numeracija su mu 4. Njegov sused b je predak pa L(e) = b.Pre = 2. Njegov sused d je roditelj. Čvor e dakle nema dece u DFS drvetu pa nije artikulaciona tačka. Vraćamo se u čvor d.
- L(e) < L(d) pa L(d) = L(e) = 2. Njegov sledeći sused je f.
- Pokrećemo DFS iz čvora f, low link i dolazna numeracija su mu 5. Njegov sused d je roditelj. Čvor f nema dece u DFS drvetu pa nije artikulaciona tačka. Vraćamo se u čvor d.
- Čvor d nema više suseda. Zbog L(e) $\geq$ d.Pre i L(f) $\geq$ d.pre čvor d je artikulaciona tačka. Vraćamo se u čvor b.
- Čvor b ima sused e, to je grana koja povezuje pretka i potomka. Pošto je L(b) = L(e) ne ažuriramo ništa. Čvor b je artikulaciona tačka jer L(e) $\geq$ b.Pre i L(d) $\geq$ b.Pre. Vraćamo se u čvor a.
- Sledeći sused je čvor c. Pokrećemo DFS iz čvora c, low link i dolazna numeracija su mu 6. On ima samo suseda a, ali mu je to roditelj pa nema dece u DFS drvetu pa nije artikulaciona tačka. Vraćamo se u čvor a i tu je kraj algoritma jer smo obišli i sve njegove susede.
- Artikulacione tačke su dakle a (jer je koren sa 2 dece u DFS drvetu), b (jer su L(e) i L(d) $\geq$ b.Pre odnosno ne mogu da se vrate do čvora a ako bi se čvor b izbacio) i čvor d (jer su L(e) i L(d) $\geq$ d.Pre odnosno ne mogu da se vrate do čvora b ako bi se izbacio čvor d)

8. Grafovi - mostovi - pojam, karakterizacija, određivanje svih mostova u grafu, primer

Most je grana grafa čijim se uklanjanjem broj komponenti povezanosti grafa povećava. Dakle, povezan graf uklanjanjem mosta postaje nepovezan. Graf može da ne sadrži mostove, a može ih imati i više od jednog.

Primer: graf bez i sa dva mosta.



Direktan način da se u neusmerenom povezanom grafu pronađu svi mostovi bi podrazumevao da izbacujemo jednu po jednu granu i za svaku proveravamo da li je graf povezan bez nje, npr. korišćenjem DFS pretrage. Složenost tog pristupa bi bila O(|E| · (|V| + |E|)).

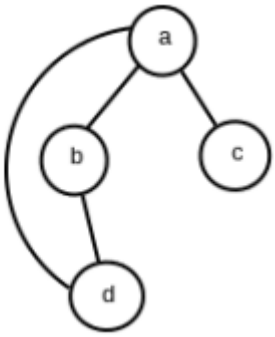
Tardžanov algoritam za traženje mostova

Važi naredno tvrđenje: grana (u, v) je most u grafu akko ne pripada nijednom ciklusu u tom grafu. S obzirom da je graf neusmeren imaće dve vrste grana u odnosu na DFS drvo - grane DFS drveta i grane koje povezuju potomka sa pretkom. Grane koje povezuju potomka sa pretkom uvek sa nekim granama DFS drveta čini ciklus pa ona ne može biti most, odnosno samo grane DFS drveta su potencijalni mostovi. Dodatno, ako je neka grana (u, v) DFS drveta most, to znači da podgraf sa korenom u čvoru v ostaje nepovezan sa delom grafa "iznad" grane (u, v), a to važi samo ako iz donjeg poddrveta ne postoji grana koja povezuje neki od čvorova tog poddrveta sa nekim pretkom iz gornjeg poddrveta. Dakle, za svaki čvor računamo koliko "visoko" možemo da se vratimo povratnim granama iz nekog od čvorova koji su deo poddrveta našeg čvora. Za ove vrednosti možemo koristiti dolaznu numeraciju pri DFS obilasku. Ova vrednost se naziva **low link** i ona će biti manja od dve vrednosti - dolazne numeracije samog čvora i najmanje vrednosti do koje se može vratiti iz nekog čvora koji je deo poddrveta našeg čvora, odnosno L(v) = min{v.Pre, $\min w.Pre$}, gde je w predak od v do kog postoji grana iz nekog čvora koji je deo poddrveta čvora v. Low link vrednost možemo efikasno računati prilikom DFS pretrage na sledeći način:

- Ako je čvor v već označen, tj. grana (u, v) povezuje potomka i pretka u odnosu na DFS drvo, onda ako je v.Pre < L(u) ažurira se L(u) na v.Pre.
- Ako je čvor v neoznačen, tj. grana (u, v) je grana DFS drveta, označava se v i njegova low link vrednost dobija istu tu vrednost, tj. L(v) = v.Pre. Nakon obrade čitavog poddrveta čvora v ako važi L(u) > L(v) ažurira se L(u) na L(v).

Grana (u, v) će biti most ako važi L(v) > u.Pre. Pošto se algoritam zasniva ne DFS pretrazi, sa konstantom ulaznom i izlaznom obradom, jasno je da će složenost biti O(|V| + |E|).

Primer: sledeći graf je zadat listom povezanosti gde su čvorovi u listama uređeni leksikografski.



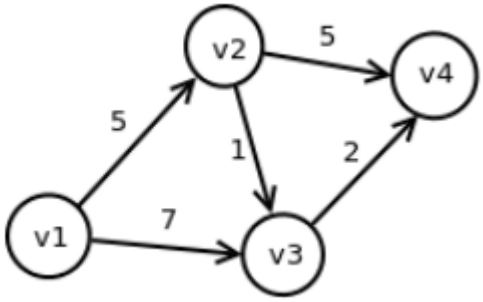
- Pokrećemo DFS iz čvora a. a.Pre = 1, L(a) = a.Pre = 1. Njegov prvi sused je b.
- Pokrećemo DFS iz čvora b. b.Pre = 2, L(b) = b.Pre = 2. Njegov prvi sused je a, ali to čini granu ka roditelju. Njegov sledeći sused je čvor d.
- Pokrećemo DFS iz čvora d. d.Pre = 3, L(d) = d.Pre = 3. Njegov prvi sused je a i to je grana ka pretku pa pošto je L(d) > a.Pre postavljamo L(d) = a.Pre. Njegov sledeći sused je b, ali to čini granu ka roditelju. Više nema suseda, vraćamo se u b.
- Važi L(d) < L(b) pa postavljamo L(b) = L(d) = 1. L(d) < b.Pre pa (b, d) nije most. Čvor b više nema suseda pa se vraćamo u a.
- Važi L(b) = L(a) pa ne ažuriramo vrednost. Sledeći sused čvora a je čvor d. Važi L(d) = L(a) pa ne radimo ništa. Sledeći sused čvora a je čvor c.
- Pokrećemo DFS iz čvora c. c.Pre = 4, L(c) = c.Pre = 4. Jedini sused je čvor a, ali to je grana ka roditelju. Vraćamo se u čvor a.
- Važi L(c) > L(a) pa ne ažuriramo vrednost. Važi L(c) > a.Pre pa (a, c) jeste most.

9. Grafovi - algoritam za određivanje najkraćih puteva iz zadatog čvora u acikličkom grafu, primer

U slučaju acikličkog grafa možemo primeniti sledeću induktivnu hipotezu: ako znamo topološki redosled čvorova, umemo da izračunamo dužine najkraćih puteva od čvora v do prvih n - 1 čvorova u tom redosledu. Dakle, iz grafa sa n čvorova uklanjamo čvor koji je poslednji u topološkom redosledu i indukcijom rešavamo problem manje dimenzije. Bazni slučaj je kada u grafu imamo jedan čvor i tada možemo reći da je dužina puta od tog čvora do samog sebe jednaka 0. Nakon što sam rešili problem manje dimenzije vraćamo čvor koji smo izbacili iz grafa nazad u graf. Pošto je to čvor koji je poslednji u poretku to znači da ni jedan drugi čvor nije dostižan iz njega, pa on neće menjati do tad izračunate najkraće puteve do ostalih čvorova. Dužinu najkraćeg puta do poslednjeg čvora ćemo sada dobiti tako što razmatramo sve čvorove w koji imaju granu do našeg poslednjeg čvora v. Potencijalni najkraći putevi će onda biti svi zbirovi najkraćeg puta do čvora w i dužine grane od w do v. Samim tim će najkraći put do v biti minimum svih tih vrednosti.

Algoritam možemo dodatno unaprediti tako što spojimo topološko sortiranje i određivanje najkraćih puteva u jedan zajednički algoritam. Induktivna hipoteza će biti da ako znamo prvih m čvorova u topološkom redosledu, umemo da izračunamo dužine najkraćih puteva od čvora v do čvorova sa rednim brojevima od 1 do m. Pamtićemo trenutne najduže puteve kako ne bismo stalno proveravali da li ka nekom čvorovu postoji grana iz nekog drugog čvora. Prilikom razmatranja čvora z, sa rednom brojem m + 1, razmatramo samo grane (z, x) koje polaze iz njega i proveravamo da li je vrednost najkraćeg puta do x manja od vrednosti najkraćeg puta do z plus dužine grane (z, x). Jasno je da ćemo onda dužine svih puteva inicijalno postaviti na $+\infty$. Dakle, umesto da grane obrađujemo "unazad", mi ih obrađujemo "unapred" kada obrađujemo njihove polazne čvorove. Ukoliko želimo i da rekonstruišemo najkraće puteve, možemo u dodatnoj strukturi čuvati roditelje čvorova na najkraćim putevima.

Primer: Odrediti najkraće puteve od čvora v_1 do ostalih čvorova.



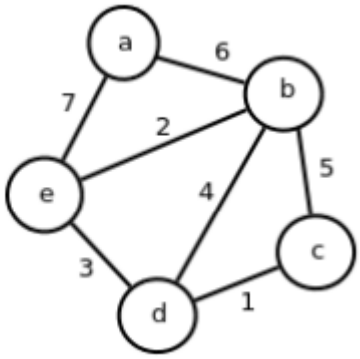
- Na početku postavljamo vrednost najkraćeg puta od čvora v_1 do čvora v_1 na 0 i to je konačna vrednost, a vrednost najkraćih puteva do ostalih čvorova na +v∞.
- Grane koje polaze iz čvora v_1 su grane do čvorova v_2 i v_3 dužina 5 i 7, pa najkraće puteve do tih čvorova postavljamo redom na min{+∞, 0 + 5} = 5 i min{+∞, 0 + 7} = 7.
- Sledeći čvor u topološkom uređenju je v_2 pa put do njega proglašavamo konačnim. Iz njega imamo grane ka čvorovima v_3 i v_4 pa ažuriramo najkraće puteve do njih redom na min{7, 5 + 1} = 6 i min{+∞, 5 + 5} = 10.
- Sledeći čvor u topološkom uređenju je v_3 pa put do njega proglašavamo konačnim. Iz njega imamo grane ka čvoru v_4 pa ažuriramo najkraći put do njega na min{10, 6 + 2} = 8.
- Sledeći čvor u topološkom uređenju je v_4 pa put do njega proglašavamo konačnim. On je poslednji čvor pa se ovde algoritam završava. Dakle, najkraći putevi od v_1 do v_2, v_3 i v_4 su redom 5, 6 i 8.

10. Grafovi - Primov algoritam 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 minimalnog povezujućem drvetu onda postoji neki drugi put do u do v pre 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

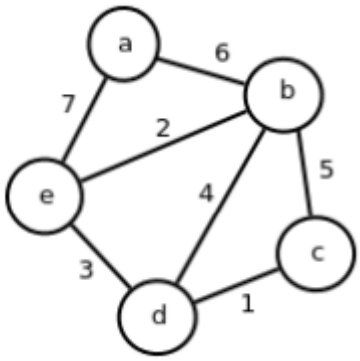


- U drvo dodajemo granu (d, c) jer je najmanje težine. Razmatramo grane (d, e), (d, b) i (c,b).
- U drvo dodajemo granu (d, e) jer je najmanje težine. Razmatramo grane (d, b), (c, b), (e, b) i (e, a).
- U drvo dodajemo granu (e, b) jer je najmanje težine. Razmatramo grane (e, a) i (b, a).
- U drvo dodajemo granu (b, a) jer je najmanje težine. Svi čvorovi su dodati - kraj algoritma.

11. Grafovi - Kruskalov algoritam za određivanje minimalnog povezujućeg drveta, primer

Za razliku od Primovog, **Kruskalov algoritam** do rešenja dolazi dodavanjem grana na trenutnu šumu, a ne trenutno drvo. Na početku, svi čvorovi grafa predstavljaju zasebno drvo, odnosno šuma sadrži |V| drveta i nijednu granu. U svakom koraku dodaje se nova grana, a samim tim se povezuju dva drveta. Nakon |V| - 1 koraka ostaće tačno jedno drvo - minimalno povezujuće drvo. Grane se razmatraju u neopadajućem redosledu. Ako najmanja grana povezuje dva čvora iz različitih drveta, dodaje se u šumu. Ako povezuje čvorove istog drveta, preskače se.

Primer



- Na početku imamo šumu sa čvorovima a, b, c, d i e i bez ijedne grane.
- Grana najmanje težine je (d, c) i ona povezuje dva drveta pa se dodaje u šumu. Čvorovi d i c sada čine jedno drvo.
- Grana najmanje težine je (e, b) i ona povezuje dva drveta pa se dodaje u šumu. Čvorovi e i b sada čine jedno drvo.
- Grana najmanje težine je (d, e) i ona povezuje čvorove dva drveta pa se dodaje u šumu. Čvorovi d, c, e i b sada čine jedno drvo.
- Grana najmanje težine je (d, b) ali ona povezuje čvorove istog drveta pa se preskače.
- Grana najmanje težine je (c, b) ali ona povezuje čvorove istog drveta pa se preskače.
- Grana najmanje težine je (b, a) i ona povezuje čvorove dva drveta pa se dodaje u pumu. Čvorovi a, b, c, d i e sada čine jedno drvo.
- Grana najmanje težine je (a, e) ali ona povezuje čvorove istog drveta pa se preskače.
- Sve grane su proverene - kraj algoritma.

12. Grafovi - Algoritam za određivanje svih najkraćih puteva u grafu indukcijom po broju čvorova i indukcijom po broju grana, složenost, primer

Algoritam zasnovan na indukciji po broju grana u grafu

Pretpostavimo da smo iz grafa uklonili granu (u, w) i da smo rešili problem na manjem grafu. Nakon dodavanja grane nazad ona može da promeni najkraći put od čvora u do čvora w. Ako važi d(u, w) < r(u, w) onda ažuriramo vrednost r(u, w) = d(u, w). Pored toga, grana može da promeni i bilo koji najkraći put od čvora v_1 do čvora v_2. Treba proveriti da li je prethodno nađeni najkraći put između v_1 i v_2 sada veći od zbira dužina najkraćeg puta od v_1 do u, dužine grane (u, w) i najkraćeg puta od w do v_2. Ako je taj zbir manji ažuriramo vrednost r(v_1, v_2). S obzirom da postoji O($|V|^2$) parova čvorova, a da graf može imati najviše O($|V|^2$) grana, ukupna složenost algoritma je O($|V|^4$).

Algoritam zasnovan na indukciji po broju grana u grafu

Pretpostavimo da smo iz grafa uklonili čvor u i da smo rešili problem na manjem grafu, odnosno našli najkraće puteve između svih ostalih čvorova. Nakon dodavanja čvora nazad u graf potrebno je izračunati najkraće puteve od čvora u do ostalih čvorova, kao i najkraće puteve od svih ostalih čvorova do čvora u. Za najkraće puteve od čvora u do ostalih čvorova razmatramo sve grane koje polaze iz čvora u. Za sve takve grane (u, v), najkraći put do čvora w će biti minimalni zbir dužine grane d(u, v) i najkraćeg puta od v do w, tj. r(v, w) što znamo da smo već izračunali. Računanje puteva od ostalih čvorova ka čvoru u se vrši na isti način. Ove provere su u najgorem slučaju složenosti $O(|V|^2)$ kad u čvor dolazi O(|V|) grana. Na kraju, potrebno je i da proverimo da li su se putevi između ostalih čvorova možda skratili ako bi išli preko u. Ova provera za svaka dva čvora je složenosti $O(|V|^2)$. Ukupna složenost je onda $O(|V|^3)$ pa je ovaj algoritam efikasniji od prvog.

13. Algebarski algoritmi - modularna aritmetika: dokazi tvrđenja koja važe

Važe sledeća tvrđenja:

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

- **Dokaz**: pretpostavimo da je $a = q_a \cdot m + r_a$ i $b = q_b \cdot m + r_b$, $0 \leq r_a, r_b < m$. Tada važi:
  $a + b = (q_a \cdot m + r_a) + (q_b \cdot m + r_b)$, odnosno $(q_a + q_b)m + r_a + r_b$
  Ako važi da je $r_a + r_b = q \cdot m + r$, $0 \leq r < m$, onda je
  $a + b = (q_a + q_b)m + r$, $0 \leq r < m$, pa je $(a + b) \bmod m = r$. Desna strana je
  $(a \bmod m + b \bmod m) \bmod m = (r_a + r_b) \bmod m = r$. □

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

- **Dokaz**: pretpostavimo da je $a = q_a \cdot m + r_a$ i $b = q_b \cdot m + r_b$, $0 \leq r_a, r_b < m$. Tada važi:
  $a \cdot b = (q_a \cdot m + r_a) \cdot (q_b \cdot m + r_b)$, odnosno $(q_a \cdot q_b \cdot m + q_a \cdot r_b + r_a \cdot q_b)m + r_a \cdot r_b$
  Ako važi da je $r_a \cdot r_b = q \cdot m + r$, $0 \leq r < m$, onda je
  $a \cdot b = (q_a \cdot q_b \cdot m + q_a \cdot r_b + r_a \cdot q_b)m + r$, $0 \leq r < m$, pa je $(a \cdot b) \bmod m = r$. Desna strana je
  $(a \bmod m \cdot b \bmod m) \bmod m = (r_a \cdot r_b) \bmod m = r$. □

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

- **Dokaz**: pretpostavimo da je $a = q_a \cdot m + r_a$ i $b = q_b \cdot m + r_b$, $0 \leq r_a, r_b < m$. Neka je $r_b - r_a + m = p \cdot m + r$, $0 \leq r < m$. Tada važi:
  $b - a = (q_b - q_a)m + (r_b - r_a) = (q_b - q_a - 1)m + (r_b - r_a + m)$, odnosno
  $(q_b - q_a - 1 + p)m + r$, pa je $(b - a) \bmod m = r$.
  Desna strana je $(b \bmod m - a \bmod m + m) \bmod m = (r_b - r_a + m) \bmod m = r$. □

14. Algebarski algoritmi - problem faktorizacije svih brojeva manjih ili jednakih od n

Ukoliko bismo koristili algoritam faktorizacije složenosti $O(\sqrt{n})$ za svaki traženi broj ukupna složenost bi bila $O(n \sqrt{n})$. Umesto toga, možemo koristiti pomoćni niz dužine n koji bi za svaki broj $k \leq n$ sadržao najmanji prost činilac broja k. Onda bismo proizvoljan broj $m \leq n$ mogli faktorisati na sledeći način: prvi prosti činilac za m uzimamo direktno iz niza, drugi prosti činilac je prvi prosti činilac broja m / p_1, treći prosti činilac je prvi prosti činilac broja m

/ ($p_1 \cdot p_2$), . . . Problem smo dakle sveli na efikasno izračunavanje najmanjeg prostog činioca svakog broja manje ili jednakog sa n i čitanja tih vrednosti iz pomoćnog niza. Ovaj niz možemo dobiti prilagođavanjem Eratostenovog sita. Na početku svaki broj postavljamo kao sopstveni najmanji prost činilac. Umesto da prilikom prolaska kroz umnoške prostih brojeva označavamo složene brojeve, mi ćemo postavljati vrednost najmanjeg prostog činioca za taj broj ako već nije postavljen. Broj m će biti prost ako je njegov najmanji prost činilac na kraju on sam. Prilikom razmatranja prostog broja d možemo izvršiti odsecanje i krenuti od d^2 jer su brojevi 2d, 3d, . . ., (d - 1)d deljivi redom sa 2, 3, . . ., d - 1 pa sigurno imaju činilac manji od d. Na kraju proste činioce broja računamo uzastopnim deljenjem broja sa njegovim najmanjim prostim činiocem.

Složenost određivanja najmanjeg prostog činioca svih brojeva do n odgovara složenosti Eratostenovog sita, tj. O($\log \log n$) ako pretpostavimo da je složenost sabiranja O(1). Složenost pronalaženja svih prostih činilaca za neki broj k bi bila O($\log k$) jer je maksimalni broj prostih činilaca $\log_2 k$ i pretpostavljamo da je složenost deljenja O(1). Složenost tog ispitivanja za sve brojeve od 1 do n bi bila $\log 1 + \log 2 + \ldots + \log n$ što je jednako $\Theta(n \log n)$. Ukupna složenost je onda O(n$\log \log n$) + O(n$\log n$) što je jednako O($n \log n$) što je efikasnije od prvog pristupa.

Primer: faktorisati sve brojeve od 1 do 50.

- Na početku je niz inicijalizovan na vrednosti od 1 do 50 tako da je svaki broj sam sebi najmanji prosti činilac
- U prvom koraku svim parnim brojevima postavljamo najmanji prost činilac na 2
- U drugom koraku svim umnošcima broja 3 počevši od vrednosti 3^2 = 9 koji nisu deljivi sa 2 postavljamo najmanji prost činilac na 3
- U sledećem koraku dolazimo do broja 4, vidimo da je njegov najmanji prost činilac 2 < 4 pa zaključujemo da je složen i preskačemo ga
- U sledećem koraku dolazimo do broja 5 i počveši od 25 svim brojevima koji nisu deljivi ni sa 2 ni sa 3 postavljamo najmanji prosti činilac na 5
- Kod broja 6 vidimo da je najmanji prost činilac 2 pa i njega preskačemo
- Kod broja 7 krećemo od 49 i svim brojevima koji nisu deljivi sa 2, 3 ili 5 postavljamo 7 za najmanji prosti činilac
- Za broj 8 vidimo da je $8^2 \geq 50$ pa je tu kraj obrade.
- Na primer, za broj 48 u nizu stoji najmanji prosti činilac 2, pa faktorizaciju svodimo na 48 / 2 = 24. Za 24 je najmanji prost činilac opet 2 pa je 24 / 2 = 12. Za 12 važi isto pa je 12 / 2 = 6. Za 6 važi isto pa je 6 / 2 = 3. Najmanji prost činilac broja 3 je on sam pa je tu kraj provere. Prosti činioci broja 48 su dakle 2, 2, 2, 2 i 3.

15. Algebarski algoritmi - problem računanja Ojlerove funkcije svih brojeva manjih ili jednakih n, primer, formulacija Ojlerove teoreme i Male Fermaove teoreme

Ako bismo za sve brojeve od 1 do n posebno računali Ojlerovu funkciju zasnovanu na faktorizaciji ukupna složenost bi bila O(n$\sqrt{n}$). Kao i u slučaju faktorizacije možemo iskoritit algoritam Eratostenovog sita i malo ga izmeniti uz angažovanje dodatnog memorijskog prostora. Iz formule za vrednost Ojlerove funkcije možemo videti da za sve brojeve deljive nekim prostim brojem p u toj vrednosti figuriše činilac 1 - $\frac{1}{p}$. Dakle, možemo redom prolaziti kroz sve proste brojeve i vrednosti Ojlerove funkcije svih umnožaka tekućeg prostog broja pomnožiti izrazom
1 - $\frac{1}{p}$. Pošto u vrednosti Ojlerove funkcije za broj n figuriše i sam broj n, jasno je da ćemo pomoćni niz na početku inicijalizovati na same brojeve od 1 do n. Za razliku od faktorizacije ovde vrednosti za p razmatramo od 2 do n jer je neophodno da prođemo sve vrednosti do n zbog prolaska kroz sve umnoške broja p i zbog postavljanja vrednosti svih prostih brojeva na p - 1. Ako u toku razmatranja broja p važi ϕ(p) = p to znači da je p prost pa vrednost postavljamo na p - 1, a potom vrednosti svih umnožaka broja p množimo sa 1 - $\frac{1}{p}$. Ponovo za razliku od

algoritma kod faktorizacije ovde ne možemo krenuti od vrednosti p^2 jer moramo sve umnoške broja p pomnožiti sa prethodnim izrazom, a to uključuje i 2p, 3p, . . ., (p - 1)p.

Primer: odrediti vrednost Ojlerove funkcije svih brojeva manjih ili jednakih 20.

- Na početku inicijalizujemo pomoćni niz na vrednosti od 1 do 20
- U prvom koraku vidimo da je vrednost Ojlerove funkcije za 2 jednaka 2 (iz niza) pa tu vrednost postavljamo ne 2 - 1 = 1. Sada za sve parne brojeve množimo vrednost Ojlerove funkcije sa 1 - $\frac{1}{2}$ = $\frac{1}{2}$
- U sledećem koraku dolazimo da broja 3 i pošto mu je vrednost Ojlerove funkcije 3 postavljamo je na 2. Prolazimo kroz sve njegove umnoške i vrednost Ojlerove funkcije im množimo sa 1 - $\frac{1}{3}$ = $\frac{2}{3}$
- Broj 4 je složen pa se preskače. Vrednost Ojlerove funkcije za 5 se postavlja na 4 i za sve njegove umnoške se vrednost Ojlerove funkcije množi sa $\frac{4}{5}$
- Broj 6 se preskače kao složen, a broju 7 se vrednost Ojlerove funkcije postavlja na 7. Zatim se svi njegovi umnošci množe sa $\frac{6}{7}$
- Brojevi 8, 9 i 10 se preskaču kao složeni, a brojevima 11, 13, 17 i 19 se postavlja vrednost Ojlerove funkcije redom na 10, 12, 16, 18. Oni nemaju umnoške manje od 20 pa tu nema dodatnih ažuriranja.

Iako u algoritmu ne idemo do $\sqrt{n}$ već do n, složenost odgovara složenosti Eratostenovog sita i iznosi O($n \log \log n$).

Ojlerova teorema: Ako je n $\geq$ 1 i a ceo broj uzajamno prost sa n, onda važi $a^{\phi(n)} \equiv 1$ (mod n). Ako u Ojlerovu teoremu umesto n uvrstimo prost broj p i ako a nije deljivo sa p onda važi $a^{p-1} \equiv 1$ (mod p). Množenjem obe strane sa a dobijamo tvrđenje Male Fermaove teoreme.
Mala Fermaova teorema: Ako je p prost broj, onda za svaki ceo broj a važi $a^p \equiv a$ (mod p). Ako je a deljivo sa p onda i a^p i a daju ostatak 0 pri deljenju sa p pa teorema važi. Ako a nije deljivo sa p teorema može da se zapiše i kao
$a^{p-1} \equiv 1$ (mod p).

16. Algebarski algoritmi - prošireni Euklidov algoritam: formulacija problema, izvođenje veza potrebnih za rekurzivni algoritam, primer

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 mod r_1, zatim ostatak r_3 = r_1 mod 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) = . . . = 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 NZD-a preko dva uzastopna člana u nizu ostataka

Pošto je r_0 = a i r_1 = b, za nzd(a, b) = d, problem možemo formulisati na sledeći način: broj d izraziti kao celobrojnu linearnu kombinaciju ostataka r_0 i r_1, odnosno d = r_k = x · r_0 + y · r_1. Ovaj problem možemo rešiti indukcijom. Bazni slučaj odgovara predstavljanju d preko dva uzastopna ostatka r_{k-2} i r_{k-1}: d = r_k = r_{k-2} - $q_{k-1} r_{k-1}$, što odgovara pretposlednjem deljenju u Euklidovom algoritmu. Korak indukcije bi bio da se polazeći od linearne kombinacije ostataka r_i i r_{i+1}, d = $x' \cdot r_i + y' \cdot r_{i+1}$, broj d izrazi kao celobrojna linearna kombinacija ostataka r_{i-1} i r

$_i$:
d = x$''$ · r_{i-1} + y$''$ · r_i. Pošto važi r_{i+1} = r_{i-1} - $q_i r_i$, zamenom dobijamo:

d = x$'$ · r_i + y$'$ · r_{i+1} = x$'$ · r_i + y$'$ · (r_{i-1} - $q_i r_i$) = y$'$ · r_{i-1} + (x$'$ - q_iy$'$) · r_i, odnosno: x$''$ = y$'$ i y$''$ = x$'$ - q_iy$'$. Indukcijom po i,
i = k - 2, k - 3, . . ., 1, možemo dobiti d kao celobrojnu linearnu kombinaciju bilo koja dva uzastopna člana niza ostataka. Specijalno za i = 0 dobija se traženi izraz. Ova verzija naziva se **prošireni Euklidov algoritam**. Ova induktivna konstrukcija je pogodna za rekurzivnu implementaciju jer nove vrednosti x i y dobijamo preko prethodnih. Složenost odgovara složenosti osnovnog Euklidovog algoritma, odnosno jednaka je O($\log(a + b)$).

Primer: odrediti brojeve x i y tako da važi 3 = 33x + 24y.

- Pošto je nzd(33, 24) = 3 možemo iskoristiti prethodni postupak. Niz jednakosti je sledeći: nzd(33, 24) = nzd(24, 9) = nzd(9, 6) = nzd(6, 3) = nzd(3, 0) = 0. Vidimo da je r_k = 3, pa je r_{k-1} = 6, a r_{k-2} = 9. Idemo unazad:
  d = 3 = 9 - 6 = 9 - (24 - 2 · 9) = 3 · 9 - 24 = 3 · (33 - 24) - 24 = 3 · 33 - 4 · 24, pa je x = 3 i y = -4.

17. Algebarski algoritmi - modularni multiplikativni inverz, izvođenje iterativnog 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 = x · a + y · m. 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. Primećujemo da nam vrednost y uopšte nije bitna pa umesto rekurzivne konstrukcije možemo koristiti iterativnu konstrukciju proširenog Euklidovog algoritma. Ona nam daje veze x_{i+1} = x_{i-1} - $q_i x_i$ i y_{i+1} = y_{i-1} - $q_i y_i$, odavde je jasno da nam y nije potrebno ni u jednom koraku da bismo računali x. Na ovaj način smo algoritam traženja multiplikativnog inverza sveli na iterativni 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 tako da važi 1 = x · 5 + y · 7. Na početku važi
  r_0 = 7 = 1 · 7 + 0 · 5 i r_1 = 5 = 0 · 7 + 1 · 5, pa su početne vrednost za x 0 i 1. Koristimo vezu x_{i+1} = x_{i-1} - $q_i x_i$ i računamo vrednosti iterativno:

- $r_0 = 7$, $r_1 = 5$, $x_0 = 0$, $x_1 = 1$, $q_1 = r_0 / r_1 = 7 / 5 = 1$, $r_2 = r_0 - q_1 r_1 = 7 - 1 \cdot 5 = 2$, $x_2 = x_0 - q_1 x_1 = 0 - 1 \cdot 1 = -1$
- $r_1 = 5$, $r_2 = 2$, $x_1 = 1$, $x_2 = -1$, $q_2 = r_1 / r_2 = 5 / 2 = 2$, $r_3 = r_1 - q_2 r_2 = 5 - 2 \cdot 2 = 1$, $x_3 = x_1 - q_2 x_2 = 1 - 2 \cdot (-1) = 3$
- $r_2 = 2$, $r_3 = 1$, $x_2 = -1$, $x_3 = 3$, $q_3 = r_2 / r_3 = 2 / 1 = 2$, $r_4 = r_2 - q_3 r_3 = 2 - 2 \cdot 1 = 0$, dobili smo ostatak 0 pa je ovde kraj algoritma, MMI = $x_3 = 3$.

18. Algebarski algoritmi - računanje proizvoda polinoma na dva načina: kada je polinom zadat nizom koeficijenata i kada je dat vrednostima na skupu tačaka

Polinom $P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$ se može predstaviti nizom svojim koeficijenata uz 1, x, x^2, . . .,
x^{n-1}, ali se može predstaviti i svojim vrednostima u n različitih tačaka i one će jednoznačno odrediti polinom. Na primer, polinom stepena 1 je jedinstveno određen sa vrednostima u dve različite tečake, a polinom stepena 2 vrednostima u tri različite tačke.

Množenje dva polinoma koji su zadati nizom svojih koeficijenata može se izvršiti osnovnim algoritmom koji podrazumeva množenje svakog monoma jednog svakim monomom drugog polinoma i on je složenosti $O(n^2)$. Može se koristiti i Karacubin algoritam zasnovan na primeni dekompozicije koji je složenosti $O(n^{\log_2 3})$. Računanje vrednosti polinoma u tački će biti složenosti O(n).

Množenje dva polinoma koji su zadati skupom tačaka je dosta jednostavnije izračunati. Ako imamo dva polinoma stepena n - 1, njihov proizvod će biti stepena 2n - 2, odnosno potrebno je 2n - 1 tačaka da bi se taj polinom jednoznačno odredio. Ako naše polinome predstavimo pomoću 2n - 1 različitih tačaka onda će množenje zahtevati samo 2n - 1 tačaka, pa će složenost biti O(n). To važi jer je vrednost polinoma PQ u tački a jednaka proizvodu vrednosti polinoma P u tački a i vrednosti polinoma Q u tački a. Računanje vrednosti polinoma u tački će biti složenosti $O(n^2)$.

reprezentacija	računanje vrednosti polinoma u tački	množenje polinoma
koeficijenti	O(n)	$O(n^2)$ ili $O(n^{\log_2 3})$
vrednosti	$O(n^2)$	O(n)

Vidimo da je kod obe reprezentacije jedna operacija efikasna, a druga nije. Zbog toga je potrebno naći način kako efikasno prebaciti polinom iz jedne u drugu reprezentaciju. Da bismo sa koeficijenata prešli na vrednosti u tačkama potrebno je da za svaku tačku izračunamo njenu vrednost (pomoću Hornerove šeme) pa je onda ukupna složenost kvadratna. Da bismo sa vrednosti u tačkama prešli na koeficijente (interpolacija) koristimo Lagranžovu interpolacionu formulu čija će složenost takođe biti kvadratna. S obzirom da ne postoji pravilo koje tačke treba da koristimo za reprezentaciju, imamo slobodu da izaberemo pogodan skup različitih n tačaka. **FFT algoritam**, odnosno **brza Furijeova transformacija**, koristi specijalan skup tačaka tako da se obe operacije mogu efikasnije izvršavati.

19. Niske - traženje uzorka u tekstu: opis problema, Rabin-Karpov algoritam i algoritam zasnovan na z-nizu, primeri, složenost ovih algoritama.

Neka su $T = t_0 t_1 \ldots t_{n-1}$ i $P = p_0 p_1 \ldots p_{m-1}$ dve niske iz konačne azbuke. Prva niska koja je po poravilu duža naziva se **tekst**, a druga **uzorak**. Problem traženja uzorka u tekstu se može definisati na sledeći način:

- Za dati tekst T = $t_0t_1 \ldots t_{n-1}$ i uzorak P = $p_0p_1 \ldots p_{m-1}$ ustanoviti da li postoji segment niske T jednak P, a ako postoji, pronaći sva njegova pojavljivanja u tekstu.
  Ovaj problem ima primenu u proveri pravopisa i gramatike, pretraživačima, analizi društvenih mreža, ali i u molekularnoj biologiji kada je potrebno pronaći neke uzorke u okviru molekula RNK ili DNK.

Rabin-Karpov algoritam problem rešava korišćenjem heširanja. Ideja algoritma je da se izračuna heš-vrednost uzorka P dužine m i svih segmenata teksta T dužine m i da se uporede njihove vrednosti. Heš-vrednost proizvoljnog segmenta teksta se može izračunati u vremenu O(1) na osnovu heš-vrednosti prefiksa teksta T i stepena broja p po modulu m, pa je izračunavanje heš-vrednosti svih segmenata složenosti O(n). Izračunavanje heš-vrednosti uzorka je složenosti O(m), dok je ukupna složenost svih poređenja O(n). Ukupna složenost je dakle O(n + m).

Algoritam zasnovan na z-nizu prvo konstruiše nisku P#T gde je P uzorak, T tekst, a # specijalan karakter koji se ne javlja ni u uzorku ni u tekstu, a potom računa vrednosti z-niza te niske. Z-niz će nam onda ukazati gde se u tekstu pojavljuje traženi uzorak, jer će na tim pozicijama z-niz imati vrednost jednaku dužini uzorka. Jasno je da vrednosti z-niza na pozicijama od 0 do m, odnosno na pozicijama uzorka, nisu bitne ali je potrebno računati ih kako bi se efikasno sproveo z-algoritam. Složenost odgovara složenosti z-algoritma, tj. linearna je u odnosu na dužinu niske pa je jednaka O(m + n).

Primer: Pronaći pojavljivanja uzorka 'ra' u tekstu 'abrakadabra'.

- Formiramo nisku 'ra#abrakadabra', a potom računamo vrednosti njenog z-niza pomoću z-algoritma.

  ```
  |0|1|2|3|4|5|6|7|8|9|10|11|12|13|
  |r|a|#|a|b|r|a|k|a|d| a| b| r| a|
  |-|0|0|0|0|2|0|0|0|0| 0| 0| 2| 0|
  ```

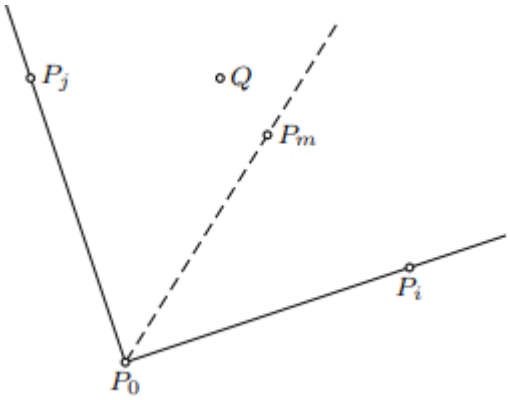
  Prolaskom kroz vrednosti z-niza vidimo da je vrednost na pozicijama 5 i 12 jednaka 2, što je dužina uzorka, što znači da se počev od tih pozicija u tekstu 'abrakadabra' javlja uzorak 'ra'.
- Kod Rabin-Karpovog algoritma bismo izračunali heš-vrednost uzorka, a zatim tehnikom prefiksnih suma redom računali heš-vrednosti svih segmenata dužine 2 u tekstu. Nakon toga se porede heš-vrednosti svih segmenata sa uzorkom. Prilikom poklapanja heš-vrednosti uzorka sa heš-vrednostima segmenata imamo opciju da verujemo heš-funkciji i ne proveravamo da li se uzorak i segment stvarno poklapaju (što ugrožava korektnost) ili možemo dodatno proveriti poklapanje karakter po karakter (što ugrožava složenost).

20. Geometrijski algoritmi - ispitivanje da li tačka pripada konveksnom mnogouglu, analiza složenosti, primer

Ukoliko su temena konveksnog mnogougla data u smeru suprotnom od smeta kazaljke na satu, da bi tačka Q pripadala mnogouglu dovoljno je proveriti da li se ona nalazi s leve strane svake usmerene stranice P_iP_{i+1} mnogougla, odnosno da li za svako i trougao $P_iP_{i+1}Q$ ima pozitivnu orijentaciju. Ova ideja predstavlja uopštenje algoritma za ispitivanje da li tačka pripada datom trouglu. Složenost algoritma je O(n).

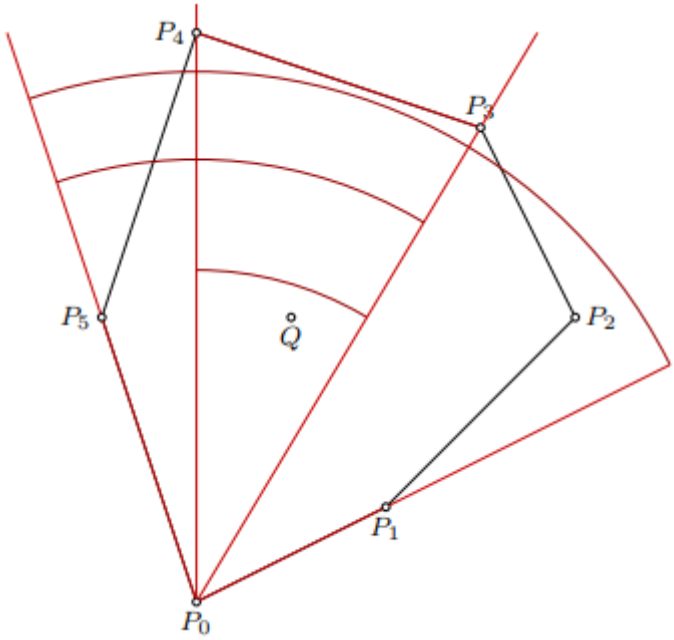
Efikasniji algoritam se oslanja na binarnu pretragu. **Lema**: Neka se tačke Q i P_m nalaze unutar ugla $\angle P_iP_0P_j$. Tada važi:

- ako su tačke P_i i Q sa iste strane prave P_0P_m, tada tačka Q pripada uglu $\angle P_iP_0P_m$
- ako su tačke P_j i Q sa iste strane prave P_0P_m, tada tačka Q pripada uglu $\angle P_mP_0P_j$
- ako tačka Q pripada pravoj P_0P_m, tada tačka Q pripada uglu $\angle P_mP_0P_j$



Binarnom pretragom tražimo vrednost i tako da tačka Q pripada uglu $\angle P_iP_0P_{i+1}$. Održavamo tekući ugao $\angle P_iP_0P_j$ kome pripada tačka Q, koji inicijalizujemo na $\angle P_1P_0P_{n-1}$. U svakom koraku za tačku P_m biramo središnju tačku, tj. tačku sa indeksom $\lfloor \frac{i+j}{2} \rfloor$ i ažuriramo ugao kome pripada tačka Q prema prethodnoj lemi. Zaustavljamo se kada tačke P_i i P_j postanu susedna temena mnogougla. Složenost algoritma je $O(\log n)$ jer se on zasniva na binarnoj pretrazi.

Primer: odrediti da li tačka Q pripada mnogouglu P.



- Ugao inicijalizujemo na $\angle P_1P_0P_5$, m = $\lfloor \frac{1+5}{2} \rfloor$ = 3 pa je P_m = 3. Q će pripadati $\angle P_3P_0P_5$. Onda je m = $\lfloor \frac{3+5}{2} \rfloor$ = 4 pa je P_m = 4. Q će pripadati $\angle P_3P_0P_4$. P_3 i P_4 su susedna temena što znači da smo pronašli ugao $\angle P_iP_0P_{i+1}$ kome pripada tačka Q, što znači da Q pripada mnogouglu

21. Geometrijski algoritmi - konstrukcija konveksnog omotača – algoritam uvijanja poklona, analiza složenosti, primer

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

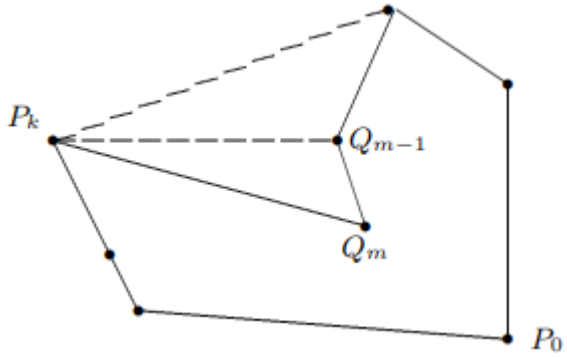
Algoritam uvijanja poklona: Krećemo od ekstremne tačke, npr. tačka sa najmanjom vrednošću x-koordinate i najmanjom vrednosti y-koordinate. Ekstremna tačka uvek pripada konveksnom omotaču. U svakom koraku dodajemo novo teme, tj. novu stranicu konveksnog omotača. Jedno teme nove stranice biće poslednja 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. Ako je m broj temena rezultujućeg omotala, a n broj tačaka skupa onda je složenost algoritma O(mn), odnosno složenost zavisi i od dimenzije ulaza i od dimenzije izlaza. Kada konveksni omotač trougao složenost će biti linearna, a kada sve tačke čine omotač složenost je kvadratna.

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

22. Geometrijski algoritmi - konstrukcija konveksnog omotača – Grejemov algoritam, primer

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

Grejemov algoritam započinje sortiranjem tačaka prema uglovima, slično kao pri konstrukciji prostog mnogougla. Od sortiranja potiče i njegova složenost O(n$\log n$). Neka je P_0 ekstremna tačka, npr. sa najvećom x-koordinatom i najmanjom y-koordinatom. Za svaku tačku P_i iz skupa računamo ugao prave P_0P_i i x-ose i sortiramo ih po veličini. Tačke prolazimo redom i pokušavamo da identifikujemo temena konveksnog omotača. Kao i kod algoritma uvijanja poklona pamtimo konveksni put sastavljen od dela prođenih tačaka pa će na kraju biti obuhvaćene sve tačke i imaćemo konstruisan konveksni omotač. Razlika u odnosu na algoritam uvijanja poklona je to što tekući konveksni put ne mora da bude deo konačnog konveksnog omotača, odnosno tekući konveksni put može sadržati neke tačke koje će kasnije biti eliminisane. Algoritam se onda zasniva na induktivnoj hipotezi: ako je dato n tačaka u ravni uređenih prema algoritmu za konstrukciju prostog mnogougla, onda umemo da konstruišemo konveksni put preko nekih od prvih k tačaka, takav da odgovarajući konveksni mnogougao obuhvata prvih k tačaka.



Slučaj k = 1 je trivijalan. Označimo konveksni put dobijen za prvih k tačaka sa P = Q_0, Q_1, . . ., Q_m. Potrebno je da proširimo hipotezu na k + 1 tačaka. Posmatrajmo ugao između pravih $Q_{m-1}Q_m$ i Q_mP_k. Ako je on manji od π (meri se iz unutrašnjosti mnogougla) onda se tačka P_k može dodati postojećem putu jer će on ostati konveksan. U protivnom, tačka Q_m leži u mnogouglu dobijenom zamenom tačke Q_m u putu P sa tačkom P_k i povezaivanjem tačke P_k sa tačkom P_0. Nakon izbacivanja tačke Q_m put ne mora uvek biti konveksan, pa moramo unazad da nastavimo sa proverama poslednje dve stranice puta, sve dok ugao između njih ne postane manji od π. Ovo će se u najgorem slučaju desiti na samom kraju kada se razmatraju tačke Q_0, Q_1 i P_k zbog prethodnog sortiranja. Ponovo, umesto računanja uglova možemo razmatrati orijentaciju tačaka. Ako je orijentacija trougla $Q_{m-1}Q_mP_k$ pozitivna, onda je ugao između pravih $Q_{m-1}Q_m$ i Q_mP_k manji od π.

Primer: pogledati Predavanje 13 od 2:31:45 do 2:40:33