Opisi algoritama

IBT 2011. MASTERS LIGA – 2. kolo

Komentare i pitanja uputite na askurdija@gmail.com *ili na* frane.kurtovic@gmail.com .

Visoravan (P1-1)

Za svaki element niza čiji je prethodnik manji od njega *for* petljom idemo u desno, te zaustavimo for petlju čim dođemo do broja različitog od trenutačnog. Ako je taj broj manji od trenutačnog, to znači da smo tom *for* petljom zapravo prešli po nekoj visoravni, te samo zapamtimo njenu duljinu i provjerimo je li veća od maksimalne. Na kraju ispišemo taj maksimum.

Ograničenje da će niz počinjati i završavati s nulom, nam garantira da s *for* petljom koja prolazi po visoravni nećemo nikada izaći iz niza, to nam pojednostavljuje kodiranje.

Zid (P1-2, P2-1)

Primijetite da horizontalnih i vertikalnih dijelova ograde ima jednako. Najlakši način za izbrojati koliko ima horizontalnih dijelova je da prebrojimo na koliko mjesta se horizontalni dio pretvara u vertikalni ili obrnuto. Ta mjesta prepoznajemo tako što ograde čine „oblik slova L“, odnosno jedna od susjednih ograda se nalazi lijevo ili desno, a druga se nalazi gore ili dolje. Rješenje je broj takvih mjesta podijeljen s 2.

Rima (P2-2)

Zadatak je poprilično izravan oko toga što treba napraviti, jedino ima par stvari na koje treba pripaziti. Prvo da bi učitali cijeli redak potrebno je koristiti u *C*-u naredbu *fgets*, a ne *scanf* jer ona učitava samo do prvog razmaka. Određivanje posljednjeg sloga u stihu je jednostavno napraviti pomoću *for* petlje koja se zaustavlja kada dođe do prvog samoglasnika ili razmaka. Ako se zaustavila na samoglasniku tada treba provjeriti i znak prije je li suglasnik (treba paziti da nije razmak na tom polju i da ta pozicija nije negativna).

Jasmin (P1-3)

Najjednostavnije rješenje je isprobati sve kombinacije koje se mogu dobiti zadanim operacijama. To radimo rekurzivno, tako da kao jedan parametar prosljeđujemo *trenutačnu riječ,* a kao drugi parametar prosljeđujemo *vrijednost* koju smo do sada skupili. Kada nam ostane jedan ili niti jedan znak u *trenutačnoj riječi,* onda u neki vanjski niz označimo koju smo vrijednost ostvarili. Primijetite da je najveći mogući broj različitih vrijednosti koje možemo skupiti *(**N**-1) ** (***N***-3) * (***N***-5) *… * (1 ili 2). Ta

vrijednost za *N = 16* iznosi 2.027.025. Primijetite da je i ukupna složenost programa jednaka ***O** (broj kombinacija)*, što je dovoljno malo da se izvrši u jednoj sekundi.

Recesija (P1-4, P2-3)

Dijkstrinim algoritmom možemo izračunati sve najkraće puteve iz glavnog grada, i onda za svaki grad uzmemo u stablo onu vezu koja je do njega dovela u najkraćem putu. Te veze sigurno čine stablo jer ih je *N-1* i ne postoji ciklus.

Kada to vrijedi, možemo za svaki grad pretpostaviti da je glavni, zatim izračunati sve najkraće puteve iz tog grada, te kao rješenje za pretpostavljeni glavni grad uzeti najveći od tih najkraćih puteva.

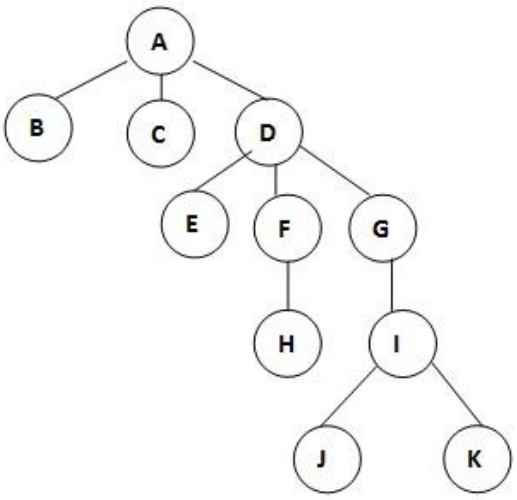
Od svih tih vrijednosti treba ispisati minimalnu.

Ovo je najjednostavnije implementirati pomoću poznatog Floyd-Warshall-ovog algoritma, koji računa za svaki par čvorova najkraći put između njih, u složenosti $O(N^3)$.

Zagrade (P1-5, P2-4)

Od svakog stabla je moguće dobiti regularan izraz zagrada,rekurzivnim algoritmom, koji je blaga modifikacija tako zvanog *preorder* obilaska stabla, samo što na početku funkcije ispišemo otvorenu zagradu, a na kraju zatvorenu.

Jedan od mogućih preorder obilazaka ovog stabla je: ABCDEFHGIJK, te će takav obilazak dati sljedeći regularan izraz zagrada: (()()(()())(((()()))))).



U zadatku je nama dan regularan izraz zagrada, te ga trebamo pretvoriti u stablo, odnosno potreban nam je obrnut postupak od ovoga.

To se opet može rekurzivno riješiti, tako da u vanjskoj varijabli pamtimo indeks u nizu zagrada, te da rekurzija kao parametar prima čvor u kojem se nalazimo. U rekurziji prvo povećamo indeks za jedan, te *while* petlju vrtimo sve dok indeks pokazuje na otvorenu zagradu, i u toj petlji svaki put napravimo novu vezu od vrha do tog vrha kojeg predstavlja ta otvorena zagrada (isti indeks nam može služiti i za označavanje vrhova).

Sad je jasno da upiti pronalaska prve zajedničke zagrade zapravo znače pronalazak najnižeg zajedničkog pretka, taj je problem poznat kao *LCA (Lowest Common Ancestor)*. Taj se algoritam u suštini sastoji od dva koraka, dva čvora se prvo dovedu na istu razinu, tako da se onaj niži dovede

gore za potreban broj mjesta. Zatim se oba dva zajednički pomiču prema gore, dok ne postanu jednaki. Naravno ako bismo ih podizali uvijek za visinu jedan, tada bi složenost algoritma bila ***O (N).***

Da bi se ubrzao algoritam, na početku se za svaki vrh izračuna ne samo njegov neposredni otac, nego i njegov otac za svaku potenciju broja 2. To nam omogućava da podizanje napravimo u logaritamskoj složenosti, što znači da i mi možemo svaki upit također riješiti u logaritamskoj složenosti.

Loptica (P2-5)

Prvo što treba primijetiti je da ne moramo u svakom upitu ponovno određivati smjer okretanja svake trake, jer je jedna kombinacija okretanja uvijek optimalna (neovisno o tome s koje koordinate bacamo lopticu).

Moguće je svaku traku zamisliti kao čvor u usmjerenom grafu, a veza ***A->B*** postoji ako loptica može pasti s trake ***A*** na traku ***B***. Konstruirani graf neće imati ciklus (jer loptica uvijek pada prema dolje), često se koristi i engleska kratica **DAG**. Zbog toga možemo jednostavnom rekurzijom izračunati koliko će loptica najviše drugih čvorova posjetiti iz tog čvora.

Konstrukcija grafa

Prvo uzlazno sortiramo trake po njihovoj *y*-koordinati. Korisno je uvesti jednu dodatnu traku koja će označavati tlo, te će biti na visini 0 i protezat će se od 0-10^9. Kada obrađivanjem dođemo do neke trake, tada želimo od već obrađenih traka naći najviše trake koje zauzimaju *x*-koordinate X_1*-1*, te X_2*+1.*

Odnosno bit će nam potrebna neka struktura koja može brzo raditi sljedeće operacije:

a) postaviti brojeve u bilo kojem intervalu na neku vrijednost

b) odgovoriti koji se broj nalazi na nekoj poziciji.

Jedna takva struktura je *tournament stablo,* a druga moguća struktura je odjeljivanje niza na uzastopne blokove, taj postupak se zove *bucketing.*

Za koju god strukturu se odlučili, svejedno moramo smanjiti prostor na kojem te strukture izvode operacije, koji je trenutačno veličine 10^9. To ćemo riješiti sažimanjem koordinata, odnosno ostavit ćemo samo one koordinate koje su nam potrebne. Za svaku traku bitne koordinate su njene rubne točke, te prve susjedne točke koje su izvan trake (X_1*-1* i X_2*+1*). Ovo smijemo napraviti zato što nam nije bitna udaljenost između svake koordinate, već samo njihov redoslijed (iako postoje i načini za rješavanje i tog problema). Prostor pretraživanja smo ovime ograničili na ***4N.***

Implementacija tournament stabla stoga radi u složenosti ***O*** *(N*log(N))*, a *bucketing* pristup rezultira ***O*** *(N*sqrt(N)).* Prvi pristup donosi sve bodove, a drugi ako se pažljivo napravi donosi također sve bodove.