

Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Marin Kišić, Adrian Satja Kurdija, Vedran Kurdija, Ivan Paljak i Paula Vidas. Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Eventualna pitanja, mišljenja ili prijedloge vezane uz zadatke uputite na askurdija@gmail.com.

Zadatak: PN

Zbrajanje svih elemenata zadanog pravokutnika (jedan po jedan) presporo je za dana ograničenja i prekoračuje vremensko ograničenje za veće primjere, ali osvaja dio testnih primjera. Više je načina da ostvarimo brže rješenje. Jedna je moguća ideja zasebno promatrati svaki redak, formulama izračunati lijevi (s_1 -ti) i desni (s_2 -ti) element danog pravokutnika u tom retku, i potom koristiti formulu za sumu aritmetičkog niza između tih elemenata. Druga ideja (ostvarena u priloženom izvornom kodu) najprije računa zbroj prvog stupca danog pravokutnika (zbrajajući njegove elemente jedan po jedan) i onda uočava da svaki sljedeći stupac ima za točno $2(r_2 - r_1 + 1)$ veći zbroj od prethodnog, jer je svaki njegov element za dva veći od elementa u prethodnom stupcu, pa prolaskom od drugog do posljednjeg stupca lako računamo njihove zbrojeve. Osobitost ovog zadatka jest i činjenica da rezultat može biti jako velik, veći od standardnih tipova podataka (npr. *long long* u jeziku C++), zbog čega moramo sami implementirati zbrajanje i množenje velikih brojeva koje pamtimo kao niz znamenaka (tzv. *bigint* ili *bignum* strukture). Napomenimo da programski jezik Python nema ograničenje na veličinu cjelobrojnog podatka pa nam ovo posljednje u tom jeziku nije potrebno, no rješenje u Pythonu zbog prirode jezika nije dovoljno brzo za sve testne primjere.

Zadatak: Intervju

Fiksirajmo neki pravac i promatrajmo sva njegova sjecišta s ostalim pravcima. Primijetimo da trokut-regija može biti određena samo dvama susjednim sjecištima na tom pravcu i trećom točkom koja je sjecište dvaju pravca koji ga sijeku u tim susjednim točkama. Ostaje nam provjeriti tvore li te tri točke trokut-regiju. Odgovor je *da* ako ne postoji neko drugo sjecište između nekih dvaju vrhova tog trokuta. (Ako postoji takvo sjecište, to znači da neki četvrti pravac prolazi kroz promatrani trokut pa on nije trokut-regija.) Navedenu provjeru lako je provesti ako za svaki pravac unaprijed izračunamo sjecišta koja na njemu leže i sortiramo ih redom (npr. po x-koordinati).

Zadatak: Imaš niz?

Primijetimo najprije da, ako je moguće konstruirati niz duljine k koji zadovoljava ograničenja iz zadatka, tada je moguće konstruirati i niz duljine $k - 1$ koji zadovoljava ta ograničenja. Ovo nas svojstvo odmah navodi na standardnu ideju, binarnim ćemo pretraživanjem pretpostaviti duljinu niza te ćemo provjeriti je li moguće konstruirati niz te duljine.

Da bismo to mogli napraviti, trebamo znati odgovoriti na dva pitanja:

1. U slučaju da ne postoji beskonačan niz, za koju duljinu smo sigurni da ne postoji rješenje?
2. Kako provjeriti postoji li niz duljine k koji zadovoljava sva ograničenja?

Odgovor na prvo pitanje je 1000. Natjecatelj do ovog zaključka može doći formalno (matematičkim dokazom) ili eksperimentalno (generirajući velik broj primjera i uočavajući ovu pravilnost). Predstaviti ćemo simpatičan dokaz da ova tvrdnja vrijedi u okviru drugog podzadatka (ograničenja oblika " $a_i \geq 1$ " ili " $a_i \leq -1$ "), a općenit dokaz ostavljamo čitatelju za vježbu.

Očito je traženi niz beskonačan ako su sva ograničenja pozitivna (" $a_i \geq 1$ ") ili ako su sva ograničenja negativna (" $a_i \leq -1$ ").

Tvrđnja: Ako postoje ograničenja oblika " $a_i \geq 1$ " i " $a_j \leq -1$ ", tada nije moguće konstruirati niz duljine $a_i + a_j$.

Dokaz. Pretpostavimo da postoji niz x duljine $a_i + a_j$ koji zadovoljava oba svojstva. Promotrimo sljedeću matricu:

$$\begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_{a_i} \\ x_2 & x_3 & x_4 & \dots & x_{a_i+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{a_j} & x_{a_j+1} & x_{a_j+2} & \dots & x_{a_i+a_j} \end{bmatrix}$$

Zbog prvog ograničenja, suma svakog retka mora biti pozitivna, pa je samim time i suma svih elemenata u matrici pozitivna. Međutim, zbog drugog ograničenja, suma svakog stupca je negativna, stoga bi i suma svih elemenata u matrici trebala biti negativna. Dolazimo do kontradikcije i zaključujemo da ne postoji niz duljine $a_i + a_j$ koji zadovoljava oba navedena ograničenja. \square

Sada kada imamo gornju granicu za potrebe binarnog pretraživanja, preostaje nam odgovoriti na drugo pitanje. Budući da se svako ograničenje odnosi na sumu nekog intervala u danom nizu, prirodno je o takvim ograničenjima razmišljati u kontekstu niza prefiks (parcijalnih) suma. Preciznije, svako od Malnarovih ograničenja možemo zapisati kao niz nejednakosti oblika $p_i - p_j \leq b$ ili $p_i - p_j \geq b$, gdje s p označavamo niz prefiks suma traženog niza. Množenjem nejednakosti drugog tipa s -1 , zadatak smo sveli na traženje rješenja sustava nejednadžbi oblika $p_i - p_j \leq w_{ij}$ u skupu cijelih brojeva.

Dakako, ovaj sustav možemo jednostavno riješiti koristeći *Bellman-Ford* algoritam za traženje najkraćih puteva u usmjerenom težinskom grafu :)

Ako vam prethodna rečenica nije nimalo intuitivna, bez brige, nije bila ni autoru zadatka kada se s njom prvi puta susreo.

Modelirajmo najprije ovaj sustav nejednakosti usmejenim težinskim grafom. Preciznije, za svaku ćemo nejednakost oblika $p_i - p_j \leq w_{ij}$ dodati brid težine w_{ij} usmjeren od čvora s oznakom v_j prema čvoru s oznakom v_i . Dodajmo također i poseban čvor s iz kojeg je prema svakom drugom čvoru usmjeren brid težine 0.

Tvrđnja: Ako u tako konstruiranom grafu postoji negativan ciklus, tada sustav nejednadžbi nema rješenje.

Dokaz. Pretpostavimo da postoji negativan ciklus duljine k između čvorova $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k \rightarrow v_1$. Neka je suma bridova u tom ciklusu jednaka $S_w < 0$. Tada imamo:

$$\begin{aligned} v_2 - v_1 &\leq w_{12} \\ v_3 - v_2 &\leq w_{23} \\ &\vdots \\ v_k - v_{k-1} &\leq w_{k-1,k} \\ v_1 - v_k &\leq w_{k1} \end{aligned}$$

Sumiramo li dane nejednakosti dobivamo $0 \leq S_w < 0$, što je kontradiktorno. \square

Tvrđnja: Ako u tako konstruiranom grafu ne postoji negativan ciklus, tada $p_i = d(s, v_i)$ zadovoljava sustav nejednadžbi, pri čemu $d(a, b)$ predstavlja najkraći put između čvorova a i b .

Dokaz. Ovo slijedi direktno iz nejednakosti trokuta. Odnosno, za svaki povezani par čvorova v_i i v_j , zbog nejednakosti trokuta vrijedi $d(s, v_i) \leq d(s, v_j) + w_{ij}$. Budući da smo varijabli p_i pridružili $d(s, v_i)$, ograničenje $p_i - p_j \leq w_{ij}$ mora biti zadovoljeno. \square

Za podzadatak koji je vrijedio polovinu bodova bilo je potrebno izvući slične zaključke. Međutim, konstruirani graf sadržavao je samo bridove težine -1 , pa je bilo moguće utvrditi postoji li negativan ciklus jednostavnim DFS obilaskom, a udaljenosti je bilo moguće pronaći koristeći BFS.

Zadatak: Čekaonica

Održavamo skup (C++ set) intervala uzastopnih slobodnih mjesta od kojih je svaki predstavljen uređenim parom lijevog i desnog sjedala. Na početku je u skupu samo interval $(1, N)$. Kad se zauzme mjesto x , u setu (korištenjem funkcije *lower bound*) pronađemo interval (a, b) kojemu to mjesto pripada, obrišemo taj interval i dodamo dva novonastala: $(a, x - 1)$ i $(x + 1, b)$. Da bismo odgovorili na traženo pitanje, održavamo i skup *kandidata* koji za svaki interval sadrži njegovo "najbolje" mjesto te udaljenosti tog mjesta do najbližeg zauzetog mjesta i do ruba. Iz tog skupa brišemo i u njega ubacujemo kandidate usporedno s brisanjem ili dodavanjem intervala u prvi skup. Kandidata za pojedini interval određujemo kao mjesto na njegovoj sredini (ako su dva takva, uzimamo "bolje"), osim ako je interval na rubu niza kada uzimamo rubno mjesto. Da bismo pronašli najbolji od svih kandidata, iz skupa kandidata dohvaćamo prvi element, tj. najmanju trojku (udaljenost do zauzetog, udaljenost do ruba, oznaka mjesta).

Zadatak: Tri

Zamislamo da vrhove grafa bojimo crvenom, bijelom i plavom bojom tako da:

- brid iz bijelog vrha može ići u vrh bilo koje boje,
- brid iz crvenog vrha može ići samo u plavi vrh,
- brid iz plavog vrha može ići samo u bijeli vrh.

Uočimo da svako takvo bojenje daje jedno moguće brisanje iz teksta zadatka. Naime, brisanjem bijelih vrhova preostaju samo bridovi iz crvenog u plave vrhove, tj. ne postoji put duljine dva. Vrijedi i obrnuto: svakom brisanju iz teksta zadatka odgovara jedno bojenje u skladu s navedenim uvjetima. Naime, vrhove koji se brišu možemo smatrati bijelima, a ostale vrhove plavima ako vode u bijeli vrh, a inače crvenima. Iz pretpostavke da ne postoji ne-bijeli put duljine dva slijedi da u ovom bojenju brid iz crvenog vrha ne može ići u crveni vrh, tj. zadovoljeni su gore navedeni uvjeti bojenja.

Sada umjesto da tražimo optimalno brisanje, tražimo optimalno bojenje u smislu minimalnog broja bijelih vrhova. Taj je problem lakše riješiti jer je, pojednostavljeno govoreći, za neki vrh dovoljno pratiti boju prethodnog vrha da bismo odlučili kako njega obojiti, a u originalnoj perspektivi morali bismo za "prethodna dva" vrha znati jesu li obrisani.

Zadani graf (pseudošuma, engl. *pseudoforest*), tj. svaka njegova komponenta povezanosti, sastoji se od jednog ciklusa u koji vode "stabla". Dinamičko programiranje reći će nam: ako smo ciklus počeli bojiti u vrhu *predstavnik* koji smo obojili bojom *boja_prvog*, a trenutačno smo u vrhu *trenutni* dok smo njegovog prethodnika u ciklusu obojili bojom *boja_prethodnog*, koji je minimalan broj bojenja u bijelo potreban da dovršimo bojenje ciklusa? Boju trenutnog vrha biramo prema uvjetima bojenja ovisno o boji prethodnog vrha, pazeći na slučaj kada se "vraćamo" u predstavnika (prvoobojeni vrh) ciklusa. Ovo je implementirano u funkciji *ciklus_dp* u priloženom izvornom kodu.

Prilikom bojenja trenutnog vrha u ciklusu, bojimo i cijelo njegovo "stablo", tj. skup vrhova koji izvan ciklusa izravno ili neizravno vode u taj vrh. Optimalno bojenje stabla također računamo (nešto jednostavnijim)

dinamičkim programiranjem koje nam daje minimalan broj bojenja u bijelo potreban da obojimo preostali dio stabla (idući prema listovima) ako smo vrh *trenutni* obojili u boju *boja*. Ovo je implementirano u funkciji *stablo_dp* u priloženom izvornom kodu.

Budući da osim optimalnog broja bijelih (brisanih) vrhova treba izračunati i broj odgovarajućih načina, svaka od navedenih *dp* funkcija vraća uređeni par (broj brisanja, broj načina), dok pomoćne funkcije *ili* i *presjek* kombiniraju (zbrajaju/množe) te uređene parove u situacijama kad moramo odabrati jedan ili drugi slučaj bojenja (funkcija *ili*), odnosno odabrati oba bojenja (funkcija *presjek*). Potonje nam treba primjerice u situaciji kad se graf sastoji od više komponenata povezanosti od kojih svaku treba zasebno riješiti te brojeve brisanja (bijelih vrhova) zbrojiti, a brojeve načina pomnožiti. U *ili* situaciji (npr. kad biramo jednu ili drugu boju za neki vrh) uzima se par koji sadrži manje brisanja, osim ako ih je jednako pa se broj načina zbraja.

Zadatak: Bitstring

Dinamičkim programiranjem računamo $ans(i)$ kao optimalan trošak za sastavljanje dijela traženog stringa od *i*-tog znaka do kraja. Da bismo ga izračunali, pokušavamo staviti svaki prefiks i sufiks pomoćnih stringova na to mjesto. Provjeru možemo li ga staviti možemo ubrzati ako unaprijed izračunamo hasheve svih prefiksa i sufiksa (kako pomoćnih tako i traženog stringa). Do rješenja koje je dovoljno brzo za sve bodove dovode nas različite optimizacije ove ideje kao što je dvosmjerna dinamika, korijenska heuristika, Aho-Corasick ili slične z-funkcije stringova, te sufiksne strukture umjesto hasheva.

(Ovaj je zadatak nastao iz 8. zadatka s ruskog natjecanja <http://neerc.ifmo.ru/school/archive/2015-2016/ru-olymp-roi-2016-day2.pdf>, a rješenje tog originalnog zadatka opisano je na <http://neerc.ifmo.ru/school/archive/2015-2016/ru-olymp-roi-2016-editorial.pdf>.)