

## 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](mailto:askurdija@gmail.com).

### Zadatak: Agenti

Informacije možemo predstaviti prirodnim brojevima od 1 do  $M'$ , gdje je  $M'$  broj sastanaka tipa A. Za svakog agenta potrebno je pratiti koje sve informacije zna. Za to možemo rabiti dvodimenzionalno polje Booleovih varijabli ili niz setova. Za sastanak tipa A, novu informaciju dodamo u set svakog sudionika ili zabilježimo u polju da on zna tu informaciju. Za sastanak tipa B, za svaki par sudionika tog sastanka, za svaku informaciju koju zna prvi sudionik zabilježimo da je zna i drugi ili je dodamo u njegov set. Na kraju je potrebno pronaći i ispisati sve agente koji znaju svih  $M'$  informacija.

### 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: Cijepljenje

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 (C++ multiset) duljina svih aktivnih intervala iz kojeg na jednostavan način dohvaćamo najveći element.

## Zadatak: Hram

Cilj nam je prebrojiti sve pravokutnike koji nemaju nule, a sadrže sve jedinice. Neka su  $r_m$  i  $r_M$  retci s najmanjim i najvećim indeksom u kojima se pojavljuje jedinica, a  $s_m$  i  $s_M$  stupci s najmanjim i najvećim indeksom u kojima se pojavljuje jedinica.

Fiksirat ćemo donji redak pravokutnika  $i$ , ali samo za retke za koje vrijedi  $i \geq r_M$ , budući da samo njima možemo obuhvatiti sve jedinice.

Neka je  $solve(i, s)$  broj pravokutnika s donjim retkom  $i$  koji obuhvaćaju sve jedinice u matrici, a počinju stupcem  $j$  za koji vrijedi  $j \geq s$ , tj. nalaze se u sufiksu retka  $i$  koji počinje stupcem  $s$ . Tada je konačan broj pravokutnika s donjim retkom  $i$  jednak  $solve(i, 1) - solve(i, s_m + 1)$ .

Preostaje nam osmisliti funkciju  $solve(i, s)$ . Za fiksiran sufiks donjeg retka pravokutnika  $i$ , promatrajmo dijelove stupaca uzastopnih jedinica ili upitnika, tj. stupaca uzastopnih ne-nula, koji završavaju u odabranom sufiksu retka  $i$ , tvoreći histogram. Da bismo izbrojili tražene pravokutnike u tom histogramu, prolazimo po njegovim stupcima s lijeva na desno i pitamo se koliko pravokutnika koji obuhvaćaju sve dosadašnje jedinice završava u trenutnom stupcu  $j$  (neka je on visine  $H$ ), točnije u polju  $(i, j)$ . Neka je broj takvih pravokutnika  $f(i, j)$ .  $f(i, j)$  pridodat ćemo na rješenje, tj. povratnu vrijednost funkcije, samo za one stupce  $j$  za koje vrijedi  $j \geq s_m$ , budući da samo njima možemo obuhvatiti sve jedinice.

Neka je  $req_h = i - r_m + 1$  najmanja moguća visina pravokutnika sa donjim retkom  $i$  koji obuhvaća sve jedinice. Da bismo izračunali  $f(i, j)$ , promotrimo prvi niži stupac histograma lijevo od  $j$ , nazovimo ga  $k$ . Možemo uzeti bilo koji pravokutnik s visinom iz intervala  $[req_h, H]$  i širinom do  $j - k$ , a njih ima  $(H - req_h + 1) \cdot (j - k)$ . Osim tih pravokutnika koji završavaju u polju  $(i, j)$ , do istog polja možemo produžiti i bilo koji traženi pravokutnik koji završava u polju  $(i, k)$ . Tada je

$$f(i, j) = f(i, k) + (H - req_h + 1) \cdot (j - k).$$

Za brz pronalazak prvog nižeg stupca  $k$  lijevo od  $j$  možemo koristiti npr. monotoni stog. Ako takav stupac ne postoji, onda je  $f(i, j) = (H - req_h + 1) \cdot (j - 0)$ .

Primijetimo da smo na različite načine u rješenje ukomponirali sve četiri granice jedinica u matrici koje su pravokutnici morali obuhvaćati,  $r_m$ ,  $r_M$ ,  $s_m$  i  $s_M$ . Za slučaj kada u matrici uopće nema jedinica, postavljamo  $r_M = s_M = 1$ ,  $r_m = R$ ,  $s_m = S$  te  $req_h = 1$ .

## Zadatak: Dva

Zamislimo 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.

*Napomena.* Priloženi kod rješava nešto složeniji zadatak (koji se javio u starijoj podskupini) gdje osim optimalnog broja bijelih (brisanih) vrhova treba izračunati i broj odgovarajućih optimalnih bojenja/brisanja. Zato 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 komponenta 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.