Opisi algoritama

IBT 2010., 6. kolo, juniori P1

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

1 Pogodak

Cijelo vrijeme pamtimo trenutni interval $[A, B]$ u kojem se traženi broj nalazi. Uvijek se najviše isplati pogađati za broj koji se nalazi na pola tog intervala - preciznije, za broj $(A+B)/2$ zaokružen naniže (ili naviše). Na taj način trenutni interval dijelimo na nova dva; pogledat ćemo koji je od dobivenih intervala veći i njega uzeti kao trenutni interval (jer će njega odabrati i Stjepan). Postupak ponavljamo dok nam ne ostane samo jedan broj u trenutnom intervalu; nakon toga ispišemo broj pokušaja.

2 Pakiranje

Pretpostavimo da je $A > B$ (ako je obrnuto, samo ih zamijenimo). Ako je broj bombona koji treba isporučiti djeljiv s A, očito se isplati uzeti samo vrećice od po A bombona. U suprotnom, uzet ćemo bar jednu vrećicu od B bombona, pa je možemo i odmah uzeti, umanjujući za B broj bombona koji još treba isporučiti. Ponovno provjeravamo je li broj bombona djeljiv s A; ako nije, ponovno uzimamo vrećicu od B bombona. Ovaj postupak ponavljamo sve dok broj bombona ne postane djeljiv sa A (i tada uzimamo dalje samo vrećice od po A bombona), ili dok ne uvidimo da se to nikada neće dogoditi. U potonjem slučaju još provjeravamo je li N djeljiv sa B - točnije, je li moguće isporučiti sve bombone u vrećicama od po B bombona, ili uopće nije moguće isporučiti bombone.

Kako ćemo uvidjeti da broj bombona nikad neće biti djeljiv s A? Oduzimajući broj B od broja N više puta, ostaci broja N pri dijeljenju s A se periodički ponavljaju. Prvi put kad se neki ostatak ponovi (a to će se dogoditi nakon najviše A oduzimanja), ponavljat će se i svi naredni, pa se ostatak 0 pri dijeljenju s A, ako se dosad nije pojavio, nikad neće pojaviti.

3 Operator

Zadatak rješavamo dinamičkim programiranjem. Neka je Z_n traženi broj. Lako je naći $Z_2 = 1$, $Z_3 = 2$ i $Z_4 = 5$; ti brojevi dani su u test primjerima iz teksta zadatka. Da bismo došli do općenitog izraza za Z_n, postavit ćemo formulu rekurzije, kojom ćemo vrijednosti od Z_n moći izračunati poznavajući vrijednosti Z_m za $m < n$.

Uočimo da bilo koji od $n-1$ operatora može doći posljednji na red u izračunu konačnog heša. Ako je taj operator k-ti, stavljamo zagradu oko prvih k članova i zagradu oko sljedećih $n-k$ članova. Zagrade unutar prvih k članova možemo rasporediti na Z_k načina, a unutar sljedećih $n-k$ članova na Z_{n-k} načina, što je ukupno $Z_k \cdot Z_{n-k}$ načina. Pribrajajući ove umnoške za svaki k dolazimo do rezultata za Z_n, pa računajući redom Z_2, Z_3, Z_4, Z_5... dolazimo i do Z_N.

4 Najveći

Potrebno je pronaći najveći pravokutnik s najviše K jedinica. Jedno od rješenja je odabrati gornji lijevi vrh pravokutnika i pronaći najveći takav pravokutnik u tablici.

U varijabli X pamtimo trenutni broj jedinica u pravokutniku. To radimo tako da postavimo širinu pravokutnika na jedan, i visinu maksimalno moguće povećamo (sve dok pravokutnik ne sadrži više od K jedinica). Zatim povećamo širinu za jedan, te povećamo X za broj jedinica iz dodanog stupca. Sada pravokutnik može sadržavati više od K jedinica, stoga smanjujemo visinu za jedan sve dok X ne postane manji ili jednak K (od X oduzimamo broj jedinica u retku koji izbacujemo). Na kraju ispišemo od svih validnih pravokutnika najveću površinu.

Složenost ovog algoritma je $O(N \cdot M \cdot (N + M))$.

5 Trokuti

Ključno je primijetiti da, ako za upit postoji trokut u kojem se tražena točka nalazi, tada ćemo uvijek moći taj trokut smjestiti na konveksnu ljusku zadanih točaka. Stoga zapravo za svaki upit moramo provjeriti nalazi li se unutar konveksne ljuske.

Algoritam koji za svaki upit prolazi po svim dužinama konveksnog poligona i provjerava da li se zadana točka nalazi s iste strane svake dužine, dobiva 60% bodova jer ima složenost $O(Q \cdot N)$.

Jedan od bržih algoritama je takozvani „sweep line“ algoritam. Prvo učitamo sve točke i sve upite, te od prvih N točaka ostavimo samo one koje leže na konveksnoj ljuski. Algoritam se temelji na tome da prolazimo pravcem paralelnim s y-osi s lijeva na desno, odnosno tim redoslijedom obrađujemo točke. Kada naiđemo na točku iz upita, moramo znati koje dužine konveksnog poligona siječe taj zamišljeni pravac (paralelan s y-osi) koji prolazi kroz tu točku iz upita. Taj pravac u svakom trenutku može sijeći ili nula ili dvije dužine konveksne ljuske. Ako taj pravac ne siječe konveksnu ljusku, tada se točka ne nalazi unutar konveksne ljuske. A ako siječe konveksnu ljusku u dvije točke, tada treba provjeriti nalazi li se naša točka s različitih strana obje dužine s konveksne ljuske. Ako je s različitih strana. tada se točka nalazi unutar ljuske. Kada naiđemo na točku konveksne ljuske, samo moramo obnoviti „otvorene“ dužine ljuske, one koje bi sijekao pravac paralelan s y-osi.

Ukupna složenost ovog algoritma je $O((N + Q) \cdot \log(N + Q))$.

Postoji još jedan algoritam. Povučemo iz najniže točke konveksne ljuske sve dijagonale. Znamo da, ako se upit nalazi lijevo od pravca kojeg čine najniža tocka i lijevi susjedni vrh konveksne ljuske, tada se upit sigurno ne nalazi unutar konveksne ljuske. Slično provjerimo i za desni susjedni vrh. Ako nije niti jedan od ovih uvjeta ispunjen, trebamo odrediti između koje dvije dijagonale se nalazi upit (dijagonale promatramo kao polupravce). Te dvije dijagonale možemo pronaći binarnim pretraživanjem. Kada smo ustvrdili koje su to dvije susjedne dijagonale, moramo provjeriti nalazi li se upit unutar trokuta koji zatvaraju najniža točka konveksne ljuske i druge dvije točke susjednih dijagonala.

Složenost ovog algoritma je $O(N \log N + Q \log N)$.

Ova dva pristupa imaju podjednaku složenost, a time i podjednaku brzinu izvođenja programa. Prednost drugog algoritma je što se na svaki upit može odgovoriti odmah, odnosno nije potrebno učitati sve podatke kao kod prvog algoritma.