AGOGE i-training
e-mail: agogeit@gmail.com
datum: 24. maj 2010. godine

Rešenja

III interno takmičenje, Niš 2010

Problem 001. Iako dinamički pristup se može učiniti najboljim, problem se rešava **greedy metodom**. Na početku dodajmo veštačke elemente $a[0] = a[n+1] = \infty$ radi lakše implementacije. Ovde ćemo iznati ideju koji možete sami pokazati da je zapravo korektna. Naime, u svakom trenutku tražimo element a_k za koji važi da je $a_{k-1} \geq a_k \leq a_{k+1}$, odnosno koji je testerast. Ukoliko je $a_{k-1} < a_{k+1}$ tada uništavamo elemente a_{k-1} i a_k, u suprotnom a_k i a_{k+1}.

Implementacija gore opisanog algoritma se može sveti na linearnu složenost ukoliko održavamo rastući podniz koji predstavlja početak niza. Kako uvek zamenjujemo samo prvi element ovde koristimo red kao strukturu. Drugi pristup bi bio da uvek uništavamo najmanji element.

Problem 010. Problem rešavamo dinamičkim programiranjem. Da bi dati podniz bio popunjen dovoljno je ispitati da li mu je suma veća od polovine zbir elemenata početnog niza i da li izbacivanjem najmanjeg elementa dobijamo manju sumu. Na početku sortirajmo niz a u nerastućem poretku: $a_1 \geq a_2 \geq \ldots \geq a_n$. Označimo sa m sumu niza a. Definišimo dva logička niza d i $mark$ na sledeći način:

$$d[i] = \begin{cases} true & \text{, ukoliko postoji podniz sa sumom } i \\ false & \text{, inače} \end{cases}$$

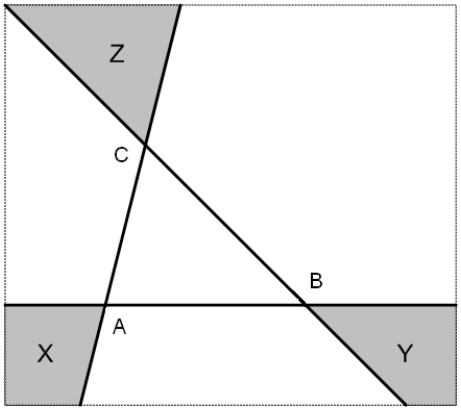
$$mark[i] = \begin{cases} true & \text{, ukoliko postoji podniz sa sumom } i \text{ koji je popunjen} \\ false & \text{, inače} \end{cases}$$

za $i \in [0, m]$. Niz d popunjavamo tako što dadajemo jedan po jedan element a po nizu se krećemo od elementa $d[m]$ pa do $d[0]$. Kada procesiramo element $a[k]$, ukoliko je $d[x] = true$ tada inicijalizujemo i element $d[x + a[k]]$ na $true$. U isto vreme, ukoliko imamo slučaj da je $x \leq s/2$ a $x + a[k] > s/2$ tada postavljamo element $mark[x + a[k]]$ na $true$. Drugim rečima, niz d nam je potreban da pravimo moguće sume podnizova, a niz $mark$ sa popunjene podnizove. Kako smo na početku elemente niza a sortirali u nerastućem poretku, uslov popunjenosti dobijamo ukoliko poslednji element koji gradi tu sumu preskače vrednost $s/2$.

Složenost algoritma je $O(n \log n + S \cdot n)$, gde je S maksimalna suma početnog niza. Sam pristup je jako sličan **problemu ranca**.

Problem 011. Jedno od očiglednih rešenja ovog problema jeste za svaki mogući trougao izračunati koliko je tačaka sadržano u njemu proverom pripadnosti za svaku tačku. Međutim, složenost ovog pristupa je $O(N^4)$ što nije dovoljno brzo.

Označimo sa $D[A][B]$ broj tačaka koje se nalaze sa desne strane usmerene duži $\overrightarrow{AB}$. Ukoliko u oblastima X, Y i Z (vidi sliku) nema tačaka, onda je broj tačaka u trouglu ABC jednak $N-(D[A][B]+D[B][C]+D[C][A])$. Međutim ukoliko to nije slučaj, broj tačaka ne možemo računati po ovoj formuli (tada bi dobili manje tačaka nego što ih zapravo ima u trouglu).



Ključno zapažanje je da ukoliko je trougao ABC rešenje zadatka, onda u oblastima X, Y i Z ne sme biti tačaka. Zaista, ukoliko bi postojala neka tačka, trougao formiran od te tačke i neka od dva temena trougla ABC bi imao više tačaka od trougla ABC, što je nemoguće. Prema tome, dovoljno je da za svake tri tačke računamo traženi broj po gornjoj formuli i da uzmemo maksimum od svih.

Složenost računanja $D[A][B]$ je $O(N^3)$, kao i računanje odgovarajućih vrednosti za svaki trougao, pa je ukupna složenost algoritma $O(N^3)$.

Problem 100. Zadatak radimo iz dva dela. Označimo sa X_h broj bodova koje možemo osvojiti ako prvi hitac ispalimo na visini h. Ukoliko uvedemo pomoćni niz $Y_h = X_h - X_{h-1}$, važi $X_h = \sum_{i=1}^{h} Y_i$. Niz Y predstavlja razliku bodova na uzastopnim visinama i ako gledamo samo jednu gomilu, možemo imati samo 4 različiti vrednosti (prilikom prelaska na konzerve različite boje). Za svaku gomilu unapred znamo pozicije prelaska i to možemo markirati tako da nam računanje nizova X i Y bude linearno po $MaxH$.

Kada znamo broj bodova za svaku visinu, zadatak nam se svodi na određivanje k-tog najmanjeg elementa u nizu visina, pri čemu u svakom koraku izbacujemo taj element. Ovo možemo uraditi korišćenjem *SegmentTree*-a u kome čuvamo niz bodova po visinama, a u unutrašnjim čvorovima čuvamo trenutni broj elemenata na odgovarajućem segmentu. Složenost algoritma je $O(M \log MaxH)$, gde je $MaxH$ najveća visina.