Descrierea Soluțiilor
Concursul Național "InfoPro", Runda 3
Grupa B

1 Problema Ghemotoace

Propunător: student Alexandru Petrescu, University of Oxford, Keble College

În primul rând considerăm că valoarea care trebuie returnată este un hash. Trebuie să observăm cum se modifică aceat hash când două elemente adiacente sunt interschimbate.

hash = $23^{n-1} * v[0] + 23^{n-2} * v[1] + ... + 23^{n-k} * v[k-1] + 23^{n-k-1} * v[k] + ... + 23^0 * v[n-1]$

Acum să presupunem că dăm swap la elementele pe pozițiile k, respectiv k + 1.

newHash = $23^{n-1} * v[0] + 23^{n-2} * v[1] + ... + 23^{n-k} * v[k] + 23^{n-k-1} * v[k-1] + ... + 23^0 * v[n-1]$

Observăm că tot ce trebuie să facem e să scădem din valoarea hashului $23^{n-k} * v[k-1]$, respectiv $23^{n-k-1} * v[k]$, iar apoi să adăugăm $23^{n-k} * v[k]$, respectiv $23^{n-k-1} * v[k-1]$.

Este recomandată precalcularea puterilor numărului 23 în funcția *init*.

Astfel nu este nevoie să parcurgem tot șirul la fiecare apel al funcției *makeSwaps* pentru a genera hash-ul.

1.1 Pentru primul test

Perechile sunt date în ordinea în care s-au efectual interschimbările deci putem pur și simplu să dăm swap la elementele din pereche și să folosim technica menționată mai sus pentru a actualiza hash-ul.

1.2 Pentru testele 2-3

Observăm că nu putem să dăm swap la elemente oricum deoarece la final nu o să rezulte ordinea corectă. Pentru a rezolva acest subtask trebuie să ne folosim de faptul că elementele din perechi sunt adiacente la un moment de timp. Deci dacă exista o pereche (x, y) după cateva swap-uri (posibil 0) cele două elemente o fie adiacente caz în care poziția lui x crește cu o unitate iar poziția lui y scade cu o

unitate (presupunând că poziția lui x este mai mică decât poziția lui y). Astfel pentru a determina noile poziții ale elementelor e nevoie să numărăm de câte ori apare fiecare într-o pereche în stânga sau în dreapta. Cum numărul de apeluri ale funcției *makeSwaps* este 1 și perechile sunt date o singură dată, un element x se află în stânga unui elemnt y din perechea (x, y) dacă $x < y$ indiferent de swap-urile realizate anterior. Astfel, pentru a rezolva acest subtask, este nevoie de o simplă parcurgere a perechilor. Pentru fiecare pereche de numere (x, y) trebuie să determinăm care element se află pe o poziție mai mică (presupunem $x < y$), scădem valorile corespunzătoare din hash, actualizăm pozițiile (pos[x] = pos[x] + 1 și pos[y] = pos[y] - 1) și adunăm noile valorile la hash.

1.3 Pentru testele 4-6

Cum valorile lui n, t, q_i sunt mici, putem să construim un algoritm similar cu *BubbleSort*. La fiecare pas parcurgem perechile de numere la care nu s-a dat swap până acum și dacă sunt adiacente le dam swap și le scoatem din listă. Repetăm procedeul până când sunt rezolvate toate perechile.

1.4 Pentru testele 7-10

Soluția pentru acest subtask este similară cu cea pentru testele $2 - 3$. Pentru aceste teste funcția *makeSwaps* este apelată de mai multe ori, la fiecare apel facându-se swap-uri pe șirul rezultat la apelul precedent. Cum o pereche (x, y) poate să apară în mai multe apeluri ale acestei funcții, condiția ca $x < y$ pentru ca x să fie în stânga lui y nu mai este adevărată. Astfel avem nevoie de un vector în care să salvăm pozițiile elementelor înainte să modificăm valoarea hash-ului. Astfel condiția de comparare se transformă în: $initialPos[x] < initialPos[y]$ unde $initialPos[x]$ reprezintă poziția elementului x în vector la apelul funcției *makeSwaps*. Soluția trebuie implementată cu grijă deoarece dacă construim vectorul *initialPos* pentru toate elementele șirului vom obține complexitatea $O(n * q)$. Pentru a obține punctaj maxim trebuie să salvăm doar pozițiile elementelor care se află în perechi. Astfel obținem complexitatea $O(n + q)$.

2 Problema Cadouri

Propunător: student Alexandru Petrescu, University of Oxford, Keble College

2.1 Subtask 1

Deoarece numărul total de cadouri este maxim 5000000, putem să reținem mărimile tuturor cadourilor într-un vector. Pentru aflarea mărimii celui de-al K-lea cadou, se poate apela funcția *std :: nth_element*() sau implementa manual o variantă a acesteia. Complexitatea este $O(\sum cnt_i)$

2.2 Subtask 2

Se pot memora m_i și cnt_i într-o structură care va fi sortată în ordine crescătoare după m_i cu un algoritm de sortare de complexitate O($NlogN$) (sau **radix sort**, care e aproape la fel de lent). Apoi, vom calcula sumele parțiale pentru cnt_i, soluția găsindu-se pe prima poziție pentru care suma parțială este mai mare sau egală cu K.

2.3 Subtask 3

Se calculează în mod repetat poziția i pentru care m_i este minim și se verifică dacă cnt_i este mai mare sau egal cu K, ceea ce înseamnă că m_i este valoarea căutată. În caz contrar, poziția i este eliminată din șir și m_i se scade din K. Complexitatea acestei soluții este O(KN).

2.4 Subtask 4

Se poate calcula numărul total de cadouri de mărime 1, 2 și 3 pentru a determina care dintre aceste valori reprezintă mărimea celui de-al K-lea cadou.

2.5 Soluție O(N) - 100 de puncte

Răspunsul ans e un număr întreg pe 32 de biți. Putem găsi, pe rând, numărul (de 16 biți) care arată primii 16 biți din (reprezentarea binară a numărului) ans, respectiv numărul (de 16 biți) care arată ultimii 16 biți din ans.

Vom calcula, mai întâi, primii (cei mai semnificații) 16 biți din răspuns, în felul următor. Calculăm, printr-o parcurgere a celor N perechi de numere date, s_j, numărul de cadouri ale căror mărimi m_i au primii 16 biți egali cu j ($0 \leq j < 2^{16}$). Apoi parcurgem șirul s de la stânga la dreapta și determinăm j-ul potrivit.

Definim $K' = K - \sum_{p=0}^{j-1} s_p$.

Pentru a găsi ultimii (cei mai puțin semnificativi) 16 biți din răspuns, e suficient să ne uităm doar la cadourile cu mărimea de așa natură încât primii 16 biți să formeze numărul j, găsit adineauri. Uitându-ne doar la ultimii 16 biți ale acestor mărimi, putem să le sortăm cu counting sort în $N + 2^{16}$ pași (echivalent, repetăm precedeul descris mai sus) și căutăm (ca înainte) al K'-lea element din șir.

Deoarece șirul dat este generat pseudo-random, pentru al doilea pas al soluției este suficientă abordarea de la subtaskul 2. Aceasta deoarece numărul de perechi (m_i, cnt_i) cu m_i având primii 16 biți egali cu j este aproximativ $\frac{N}{2^{16}}$.

2.6 Soluție alternativă O(N) - 100 de puncte

Se va implementa o funcție asemănătoare cu cea de la subtaskul 2, care în loc să afle valoarea de pe poziția K, trebuie să afle valoarea m_i minimă pentru care numărul de cadouri cu mărimea mai mică sau egală ca m_i este mai mare sau egal cu K. Algoritmul este asemănător cu Quicksort, însă se vor sorta doar anumite

porțiuni ale șirului. La fiecare pas, se alege o poziție pivot p și se împarte șirul în două șiruri, unul care conține elementele cu m_i mai mic sau egal ca m_p și unul care conține elementele cu m_i mai mare decât m_p. Această împărțire a șirului se poate realiza în timp liniar.

În cazul în care suma cnt_i pentru elementele din primul șir este mai mare sau egală cu K, înseamnă că trebuie să continuăm căutarea în acest șir, iar altfel, în al doilea șir. În cazul în care toate elementele au aceeași valoare m_i înseamnă că am găsit soluția. Acest algoritm are complexitate medie O(N) (la fiecare pas șirul se împarte în două jumătăți nu neapărat egale, dar suficient de avantajos proporționate), însă pe cazul defavorabil poate ajunge până la O(N^2) (pivotul are m_p maxim/minim la fiecare alegere). Pentru a ne asigura că anumite date de intrare nu produc acest caz, pivotul trebuie ales aleator. La fel ca la subtaskul 2, m_i și cnt_i se pot memora într-o structură pentru a simplifica partiționarea șirului.

3 Problema Media

Propunător: student Alexandru Petrescu, University of Oxford, Keble College

3.1 Pentru primul test

În primul rând observăm că pentru o subsecvență $[i, j]$, este mai ușor să comparăm M^{j-i+1} cu $\prod_{k=i}^{j} V[k]$ decât M cu $\sqrt[j-i+1]{\prod_{k=i}^{j} V[k]}$.

Cum N este maxim 50 putem să folosim două structuri repetitive de tip for pentru a parcurge toate subsecvențele posibile și a calcula produsul numerelor pe care le conțin. Produsul elementelor unei subsecvențe poate să depășească valoarea maximă care poate fi reținută în *long long*. Putem să facem un artificiu matematic și să calculăm M^{j-i+1} și $\prod_{k=i}^{j} V[k]$ modulo MOD unde MOD este un număr prim foarte mare care poate fi reținut într-o variabilă de tip int i.e. $10^9 + 7, 10^9 + 9$. Problema acum este că există o probabilitate neneglijabilă ca două numere să dea același rest la împărțirea cu MOD dar ele să nu fie egale. Există o modalitate de a reduce această probabilitate aproape de 0 astfel: Alegem două numere prime mari $MOD1, MOD2$ și verificăm dacă $M^{j-i+1}\%MOD1$ este egal cu $(\prod_{k=i}^{j} V[k])\%MOD1$ și dacă $M^{j-i+1}\%MOD2$ este egal cu $(\prod_{k=i}^{j} V[k])\%MOD2$.

3.2 Pentru testele 2-6

În primul rând să ne gândim cum rezolvăm problema pentru media aritmetică. O observație foarte importantă este următoarea:

Dacă considerăm trei indici a, b, c astfel încat $a < b < c$ și media aritmetică a numerelor $V[a...b)$ este M atunci media aritmetică a numerelor $V[a...c]$ este M dacă și numai dacă media artimetică a numerelor $V[b...c]$ este M.

Prin urmare, toate subsecvențele cu media M sunt fie indivizibile (minimale), fie compuse din mai multe astfel de subsecvențe atașate una după alta.

Fie P șirul sumelor parțiale ale lui V și Q definit astfel: $Q[i] = P[i] - M * i$ pentru orice i din intervalul $[0, N]$. Acum să analizăm ce relație obținem pentru o secvență $[i, j]$ care are media aritmetică a termenilor egală cu M.

$\frac{P[j]-P[i-1]}{j-i+1} = M$

$P[j] - P[i-1] = M * (j - i + 1)$

$P[j] - P[i-1] = M * j - M * (i - 1)$

$P[j] - M * j = P[i-1] - M * (i - 1)$

$Q[j] = Q[i-1]$

Deci o subsecvență $[i, j]$ are media aritmetică egală cu M dacă și numai dacă $Q[j]$ este egal cu $Q[i-1]$. Pentru a obține rezultatul trebuie să generăm vectorul Q iar apoi să sortăm elementele acestuia. Acum pentru fiecare subsecvență maximală de elemente egale adunăm la răspuns $\frac{L*(L-1)}{2}$ unde L este lungimea subsecvenței.

Revenind la varianta cu media geometrică.

Observăm că nu are rost să luăm în considerare subsecvențe care au elemnte cu factori primi care nu sunt divizori ai lui M. În continuare orice subsecvență la care fac referință este relevantă pentru calculul răspunsului. Acum să observăm ce se întâmplă dacă logaritmăm ecuația $M = \sqrt[j-i+1]{\prod_{k=i}^{j} V[k]}$.

$\log_2 M = \frac{\sum_{k=i}^{j} \log_2(V[k])}{j-i+1}$

Observăm că dacă logaritmăm M și elementele vectorului, problema se transformă în a găsi toate subsecvențele care au media aritmetică egală cu $\log_2 M$.

Cum M și elementele vectorului sunt puteri ale lui 2 putem să folosim 2 ca bază a logaritmului pentru a evita numerele reale.

3.3 Pentru testele 7-10

Pentru a rezolva aceste teste trebuie să gasim o modalitate de a evita numerele reale. În primul rând putem să factorizăm fiecare element din vector. Să notăm cu T numărul de factori primi ai lui M. Cum M este maxim 10^9, $T \leq 9$. Pentru calculul răspunsului sunt relevante doar subsecvențele care conțin elemente ale căror factori primi se regăsesc printre cei ai lui M. Acum putem să izolăm fiecare factor prim B din șir și din M, să logaritmăm în baza B elementele obținute dupa izolare și să calculăm vectorul Q_B specific lui B.

Exemplu:

V = [1, 4, 6, 12, 3]

Dacă vrem să izolăm numărul prim 2 vom obține:

$V_2 = [1, 4, 2, 4, 1]$

Acum nu este corect să modificăm răspunsul doar în funcție de valorile din vectorul Q_B actual deoarece dacă factorul B apare și în M și în media geometrică, nu ne garantează și că factorul B' apare și el în ambele valori. Astfel trebuie să reținem pentru fiecare poziție ce valori obținem în Q pentru fiecare

factor prim (sunt maxim 9 factori primi). O subsecventă $[i, j]$ are media geometrică M doar dacă toate valorile din $Q[j]$ sunt egale cu corespondentele din $Q[i-1]$.

Practic, e neovie să sortăm lexicografic vectorii (de câte T elemente) $Q[i]$. Complexitatea este $O(NTlogN)$.

Echipa care a pregătit setul din probleme pentru această rundă a fost formată din:

- studentă Alexandra - Maria Udriștoiu, Facultatea de Matematică și Informatică, Universitatea din București.
- student Alexandru Petrescu, University of Oxford, Keble College.
- student Adrian-Emanuel Dicu, Facultatea de Automatică și Calculatoare, Universitatea Politehnica din București.
- student Mihai-Cristian Popescu, Facultatea de Matematică și Informatică, Universitatea Babeș-Bolyai din Cluj-Napoca.
- student Denis-Andrei Banu, Facultatea de Informatică, Universitatea Alexandru Ioan Cuza din Iași
- profesor Ionel-Vasile Piț-Rada, Colegiul Național "Traian", Drobeta-Turnu Severin.
- profesor Ciprian Chește, Liceul Tehnologic "Grigore C. Moisil", Buzău.
- profesor Marius Nicoli, Colegiul Național "Frații Buzești", Craiova.
- profesor Eugen Nodea, Colegiul Național "Tudor Vladimirescu", Târgu-Jiu.
- profesor Dan Pracsiu, Liceul Teoretic "Emil Racoviță", Vaslui.