# Bucuresti, Steaua Bucuresti (Steaua)

Author: Alin-Gabriel Raileanu

Developer: Alin-Gabriel Raileanu

## Solution

Vom vedea punctajele obtinute de Steaua in ultimii $N$ ani ca fiind un vector $A$, cu $N$ elemente, indexate de la 1 la $N$.

Initial, indiferent de soluia abordata, este necesara construirea a 2 vectori: $st$ si $dr$, cu proprietatile:

- $dr[i]$ $(1 \leq i \leq N)$ : cel mai mic indice $j$ $(j > i)$ cu $A[j] > A[i]$ ($N + 1$ in caz ca nu exista);
- $st[i]$ $(1 \leq i \leq N)$ : cel mai mare indice $j$ $(j < i)$ cu $A[j] > A[i]$ (0 in caz ca nu exista).

Aceasta parte din solutie poate fi rezolvata in $O(N)$ utilizand o tehnica clasica cu stiva.

Acum, pentru fiecare indice $i$ $(1 \leq i \leq N)$, trebuie sa gasim indicele minim $j$ $(j \geq i)$ astfel incat secventa $(i, j)$ din vectorul $A$ are $K_1$ schimbari de maxim de la stanga la dreapta.

Pentru aceasta parte putem utiliza 2 solutii diferite:

- Solutie $O(N \cdot log(K_1))$ : Putem construi un tablou bidimensional $bl$, definit astfel:
  - bl[i][pw] : indicele minim astfel incat secventa $(i, bl[i][pw])$ are $2^{pw} + 1$ schimbari de maxim de la stanga la dreapta;
  - initializare: $bl[i][0] = dr[i]$;
  - recurenta: $bl[i][pw] = bl[bl[i][pw - 1]][pw - 1](1 \leq pw \leq log(K_1))$.

  Atat complexitatea timp pentru construire, cat si complexitatea spatiu sunt $O(N \cdot log(K_1))$.

  In continuare, pentru a afla rezultatele dorite, putem aplica o tehnica de tipul **binary lifting**.

  Pentru fiecare $i$ vom merge descrescator prin puteri de 2 in tabloul $bl$, schimband indicele la fiecare pas, pana cand suma acestor puteri este egal cu valoarea $K_1$.

  Cum acest proces poate fi implementat in $O(log(K_1))$ si va fi aplicat pentru fiecare indice de la 1 la $N$, complexitatea finala va fi $O(N \cdot log(K_1))$.

- Solutie $O(N)$ : Putem construi un arbore cu $N+1$ noduri, numerotate de la 1 la $N+1$, si radacina in nodul $N + 1$, astfel:
  - parintele fiecarul nod $i$ $(1 \leq i \leq N)$ va fi nodul $dr[i]$.

  Acum, putem observa ca rezultatul cautat pentru fiecare indice $i$ $(1 \leq i \leq N)$ este echivalent cu al $K_1$-lea stramos al nodului $i$ in acest arbore.

  Acest proces poate fi rezolvat in $O(N)$ pentru toate nodurile, retinand in parcurgerea DFS a arborelui lantul de la radacina la nodul curent intr-o stiva.

Analog, trebuie sa gasim si indicele maxim $k$ astfel incat secventa $(k, i)$ are $K_2$ schimbari de maxim de la dreapta la stanga. In acest caz se poate aplica una dintre solutiile de la mai sus pe vectorul $A$ inversat.

Scopul acestor rezultate este calcularea tablourilor $kst$ si $kdr$, cu proprietatile:

- $kst[i]$ $(1 \leq i \leq N)$ : cel mai mic indice $j$ $(1 \leq j \leq i)$, cu proprietatea ca $\max_{k=j}^{i} A[k] = A[i]$, astfel incat secventa $(j, i)$ are $K_1$ schimbari de maxim de la stanga la dreapta. ($N+1$ in caz ca nu exista);
- $kdr[i]$ $(1 \leq i \leq N)$ : cel mai mare indice $j$ $(i \leq j \leq N)$, cu proprietatea ca $\max_{k=i}^{j} A[k] = A[i]$, astfel incat secventa $(i, j)$ are $K_2$ schimbari de maxim de la dreapta la stanga. (0 in caz ca nu exista).

Pentru subtask-ul in care numerele sunt distincte, aceasta parte se poate calcula direct din procedeul explicat mai sus.

Totusi, daca numerele nu sunt distincte, acest aspect nu duce in toate cazurile la solutia corecta.

In acest caz, in plus, este necesar sa transmitem valorile din $kst$ si $kdr$ in urmatorul mod:

- pentru fiecare $i$ $(1 \leq i \leq N)$ vom transmite rezultatul din $kst[i]$ catre toate pozitiile $j$ $(i \leq j < dr[i])$ cu proprietatea ca $A[i] = A[j]$;
- pentru fiecare $i$ $(1 \leq i \leq N)$ vom transmite rezultatul din $kdr[i]$ catre toate pozitiile $j$ $(st[i] < j \leq i)$ cu proprietatea ca $A[i] = A[j]$.

Acest proces poate fi rezolvat in complexitate timp $O(N \cdot log(N))$ utilizand smenul lui Mars si set-uri sau in complexitate timp $O(N)$ cu observatia ca este necesara transmiterea doar catre indicele cel mai departat dintre cele alese in fiecare directie.

Raspunsul problemei va fi:

$$\max_{i=1}^{N} (kdr[i] - kst[i] + 1).$$

Complexitate timp: $O(N)$ sau $O(N \cdot log(N))$ in functie de implementare.

Complexitate memorie: $O(N)$ sau $O(N \cdot log(N))$ in functie de implementare.

# Omogen (omogen)

Author: Alexandru Gheorghies

Developer: Stefan Dascalescu

## Solution

Pentru obinerea punctajelor din primul subtask, putem folosi diverse metode brute-force care calculeaz cel mai mare divizor comun pentru fiecare pereche de valori.

O proprietate foarte important a celui mai mare divizor comun este aceea c dac $cmmdc(a, b) = 1$, atunci cele dou numere nu au niciun factor prim comun. Dac extindem aceast proprietate pentru o întreag subsecven de valori, atunci putem trage concluzia c oricare ar fi un numr prim $x$, nu poate s apar în reprezentarea în factori primi a mai mult de un numr din acea subsecven, deoarece în caz contrar, ar exista o pereche de tip $(a_2, b_2)$ cu cmmdc-ul diferit de 1.

Astfel, vom precalcula toate numerele prime mai mici decât $10^7$, precum i descompunerea în factori primi, pe care o vom stoca într-un mod compresat pentru a evita folosirea unei cantiti prea mari de memorie. Acest lucru se poate face folosind un algoritm similar cu ciurul lui Eratostene. Pentru diverse punctaje pariale se pot folosi algoritmi care afl divizorii în mod obinuit.

În final, putem folosi un algoritm de tip two-pointers folosind un vector de frecven care pstreaz informaii despre fiecare divizor prim care apare, iar atunci când avem un numr prim care apare pentru a doua oar în ir, vom scoate valori din captul din stânga pân când aceast proprietate nu mai este adevrat.

In order to obtain the scores from the first subtask, we can rely on various brute force methods which compute the greatest common divisor for every pair of numbers in the subarray.

A very important property of the greatest common divisor is that if $gcd(a, b) = 1$, then $a$ and $b$ don't have any common prime factors. If we expand this property for an entire subarray, we can conclude that for every prime $x$ which is present in the prime factorization of at least one of the numbers, then it can't show up in more than one number's factorization, because if this property wouldn't be true, there would be a pair $(a_2, b_2)$ such that its GCD would be greater than 1.

Thus, we will precompute all prime numbers less than $10^7$, together with the prime factorization which will be stored in a compressed manner in order to avoid using way too much memory. This can be done using an algorithm similar to the sieve of Eratosthenes.

Last but not least, in order to compute the number of homogenous subarrays, we will rely on the information found previously in order to track whether each prime number shows up or not using two pointers and a frequency array, and as we find a prime number which shows up twice, we will remove integers from the left.