Editorial RoAlgo PreOJI 2025

1-8 Martie 2025

Cuprins

1 Multumiri

Acest concurs nu ar fi putut avea loc fără următoarele persoane:

- Raul Ardelean, Ștefan Dăscălescu, Luca Mureșan, Stefan Vilcescu, autorii problemelor și laureați la concursurile de informatică și membri activi ai comunității RoAlgo;
- Alex Vasiluță, fondatorul și dezvoltatorul principal al Kilonova;
- Ștefan Alecu, creatorul acestui șablon LaTeXpe care îl folosim;
- Ștefan Alexandru Nuță, Ștefan Neagu, Raul Ardelean, Vlad Munteanu, Stefan Vilcescu, Tudor Iacob, Susan, Traian Danciu, testerii concursului, care au dat numeroase sugestii și sfaturi utile pentru buna desfășurare a rundei;
- Ștefan Dăscălescu, Andrei Iorgulescu și Luca Mureșan, coordonatorii rundelor;
- Comunității RoAlgo, pentru participarea la acest concurs.

2 Tomy

Autor: Raul Ardelean

2.1 Soluția oficială

Această problemă are mai multe soluții posibile, aici vom prezenta doar una dintre ele.
Vom putea începe prin a sorta șirul, iar pentru fiecare valoare de la cea maximă la cea minimă vom număra numărul de subșiruri care o au drept valoare mediană. Dacă considerăm că elementul curent are L valori mai mici decât el și R valori mai mari decât el, va trebui să numărăm câte moduri sunt de a lua un număr egal de valori la stânga și dreapta sau cel mult o valoare în plus la dreapta.
Dacă fixăm numărul de valori de la stânga, vom putea calcula în total pentru numerele din șir răspunsurile în $O(n^2)$.
Pentru a putea optimiza acest calcul, putem observa că cel mai mare număr va apărea în C_n^0 subșiruri, al doilea cel mai mare va apărea în C_n^1 subșiruri etc.
Pentru a putea calcula combinările, putem folosi fie triunghiul lui Pascal, fie precalcularea factorialelor și ale inverselor modulare.

2.1.1 Cod sursă

Soluție de 100

3 Par Impar

Autor: Mureșan Luca Valentin

3.1 Soluție 64 puncte

Observăm că la fiecare pas că lungimea șirului a se înjumătățește, așadar după cel mult $log_2(N)$ operații, șirul a va deveni gol, deci procesul se va termina.
Putem observa că operațiile sunt similare cu un algoritm de tip Divide et Impera, diferența fiind că deobicei alegem să împărțim șirul în două, prima jumătate și a două jumatate. În schimb, de data aceasta împărțim în funcție de paritate.
Așadar, putem pur și simplu la fiecare pas să alegem dacă facem prima sau a doua operație, în total vom avea cel mult $2^{log_2(N)} = N$ secvențe de operații. Dacă pentru fiecare dintre ele calculam în $O(N)$ subsecvența de sumă maximă, obținem complexitatea $O(N^2)$ pentru a genera toate secvențele de operații posibile. Vom memora toate sumele obținute într-un vector. Pentru a răspunde la întrebări, putem pur și simplu să iterăm prin toate sumele și să verificăm dacă se află între L și R.

3.2 Soluție 100 puncte

Dezavantajul primei soluții este că pentru fiecare secvență de operații în parte, calculăm subsecvența de sumă maximă. Vrem să optimizăm acest lucru. Să zicem că pentru calcularea subsecvenței de sumă maximă folosim un algoritm clasic (de exemplu, algoritmul lui Kadane). De fiecare dată când împărțim șirul în două bucăți, observăm că din valorile puse până acum în șirul b ne interesează doar două valori. Subsecvența maximă de până acum și sufixul de sumă maximă. Așadar, putem transmite aceste două informații mai departe, nefiind necesar să mai recalculăm de fiecare dată subsecvența de sumă maximă.
Complexitatea este $O(N \cdot log_2(N))$, similar cu cea de la Divide et Impera.
Rămâne să răspundem eficient la întrebări. Similar, cu soluția precedentă, ne putem ține un șir cu toate valorile. De asemenea, dacă sortăm acest șir, vom putea aplica căutare binară la fiecare întrebare pentru a afla câte valori sunt de la L la R.
Astfel, complexitatea finală este $O((N + Q) \cdot log_2(N))$.
Pentru a ne fi mai ușoară implementarea, este recomandată folosirea vectorilor STL, deși acest lucru nu este neapărat necesar.

3.2.1 Coduri sursă

100 puncte

4 Mod

Autor: Stefan Dascalescu

4.1 Soluția oficială

La prima vedere, o soluție care vine ușor în minte este aceea că vom procesa actualizările folosind o metodă de tip brute-force, sortând șirul și aplicând operația pe cele mai mari x valori. Totuși, această soluție este mult prea înceată și nu va putea rula suficient de rapid.
Pentru subtaskurile în care valorile din șir sunt mici, ne putem gândi să memorăm folosind liste pozițiile în care apar valorile de la 1 la 10^6, iar atunci când procesăm o actualizare, să parcurgem doar listele care nu sunt goale și să prelucrăm elementele în acest mod. Deși această abordare este una mai bună, nu va putea trece de subtaskurile generalizate din cauza lipsei de memorie.
Observația cheie pe care o vom folosi pentru a rezolva această problemă este aceea că dacă o valoare va suferi modificări ca urmare a unei operații de MOD, acea valoare va fi cel mult egală cu jumătate din cât era inițial.
Pentru a demonstra acest lucru, vom presupune că x va deveni egal cu $x\%y$. Noi deja știm că $x \geq b$, deci x va deveni mai mic. În primul rând, dacă $2 \cdot b \leq x$, această proprietate este respectată în mod evident. Dacă $2 \cdot b \geq x$, atunci x va scădea cu o valoare mai mare decât jumătatea sa, ceea ce face relația din nou adevărată.

Astfel, putem conclude că un număr poate ajunge la 0 din cel mult $\log x$ operații care chiar ajung să fie aplicate, ceea ce face numărul de actualizări relevante să fie proporțional cu $n \log n$. Folosind o structură de date de tip set, acest algoritm poate fi simulat folosind brute force în $O(n \log^2 n)$, ceea ce este suficient de rapid pentru a obține 100 de puncte.

4.1.1 Cod sursă

Soluție de 100