Tabăra de pregătire a lotului național de informatică - juniori, Zalău 22-27 mai
Baraj 3
Descrierea soluțiilor

Comisia științifică

May 24, 2025

Problema 1. Allp

Propusă de: Stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași

Observație

Un șir are proprietatea P dacă există cel mult un număr cu frecvență impară. Acest lucru se datorează faptului că un palindrom de lungime pară nu poate avea un număr cu frecvență impară, iar un palindrom de lungime impară are un singur număr cu frecvență impară, cel din mijloc.

Soluția 1

Putem considera pe rând fiecare submulțime de indici din intervalul $[l, r]$ al unui query, numărând câte astfel de submulțimi respectă proprietatea P în forma rescrisă mai sus. Complexitatea acestei soluții este $O(Q \cdot (r-l) \cdot 2^{r-l+1})$, care este suficientă pentru primul subtask. Ea poate fi optimizată la $O(Q \cdot 2^{r-l+1})$ dacă facem trecerea între două submulțimi în $O(1)$.

Soluția 2

De acum vom considera numerele ca fiind „normalizate”, adică vom aduce toate numerele în intervalul $[1, N]$ și vom nota cu $VALMAX$ valoarea maximă din șir. Vom procesa query-urile pe rând și pentru fiecare vom determina frecvențele numerelor care apar în respectivul interval. Fie f_x frecvența numărului x. Numărul de moduri de a selecta un număr **par** de elemente cu valoarea x (dintre cele f_x disponibile) este egal cu

$$C_{f_x}^0 + C_{f_x}^2 + C_{f_x}^4 + \ldots = 2^{f_x - 1}$$

Similar, numărul de moduri de a selecta un număr **impar** de elemente cu valoarea x este

$$C_{f_x}^1 + C_{f_x}^3 + C_{f_x}^5 + \ldots = 2^{f_x - 1}$$

Aceste formule sînt consacrate, dar pot fi și deduse prin inducție sau chiar empiric, pe cazuri particulare de coeficienți binomiali ca (1, 4, 6, 4, 1) sau (1, 5, 10, 10, 5, 1).

Prin urmare, dacă selectăm un număr pentru a avea frecvență impară, sau dacă selectăm ca toate să aibă frecvențe pare, numărul de moduri de a obține o astfel de configurație este $\prod_x 2^{f_x - 1}$. Deci, răspunsul este $[\prod_x 2^{f_x - 1} \cdot (nrdif + 1)] - 1$, unde $nrdif$ este numărul de numere distincte (adică cu frecvența nenulă) din intervalul de query. Astfel, putem obține o complexitate $O(VALMAX * Q)$ care este suficientă pentru primele 4 subtask-uri.

Soluția 3

Observăm că $\prod_x 2^{f_x - 1} = 2^{r-l+1-nrdif}$. Deci, formula finală este $2^{r-l+1-nrdif} \cdot (nrdif + 1) - 1$. Deci, pentru fiecare interval trebuie să calculăm câte numere au frecvența mai mare decât 0. Acest lucru se poate face ușor folosind algoritmul lui Mo, obținându-se astfel o complexitate de $O((N+Q) \cdot \sqrt{N})$. Această soluție se încadrează în timp pe toate subtask-urile în afară de ultimul. Ea poate fi împinsă aproape de 100p cu optimizări ca: citirea rapidă; eliminarea operațiilor modulo; precalcularea puterilor lui 2.

Soluția 4

Pentru a optimiza soluția precedentă, vom grupa query-urile după capătul lor dreapta, iterând apoi prin pozițiile de la 1 la N și menținând într-un arbore indexat binar/arbore de intervale valoarea 1 pe pozițiile în în care se află ultima apariție a unui număr până la acel moment, respectiv 0 în rest. Astfel, numărul de numere distincte dintr-un interval se obține printr-o singură interogare de sumă pe interval în structura arborescentă. Acest articol detaliază algoritmul.

Complexitate finală: $O((N + Q) \cdot \log N)$.

Problema 2. Powtop

Propusă de: prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil", Buzău

Soluția 1

Se poate demonstra că având un șir S_i, $1 \leq i \leq N$ de numere naturale pe care îl transformăm succesiv de $N - 1$ ori într-un alt șir format din **suma** oricăror 2 termeni consecutivi ai șirului precedent, numărul obținut la final poate fi calculat cu ajutorul unei expresii combinatoriale fără a mai fi nevoie să aplicăm succesiv cele $N - 1$ transformări. De exemplu pentru șirul [10, 20, 30] se obține succesiunea:

$$[10, 20, 30] \rightarrow [30, 50] \rightarrow [80]$$

Așadar

$$\begin{aligned} 80 &= 30 + 50 \\ &= 10 + 20 \cdot 2 + 30 \\ &= 10 \cdot C_2^0 + 20 \cdot C_2^1 + 30 \cdot C_2^2 \end{aligned}$$

Prin inducție matematică se poate demonstra că numărul TOP din finalul aplicării algoritmului este egal cu:

$$TOP = S_1 \cdot C_{n-1}^0 + S_2 \cdot C_{n-1}^1 + \cdots + S_n \cdot C_{n-1}^{n-1} \tag{1}$$

Să analizăm ce se întâmplă când schimbăm operația de **adunare** cu aceea de **înmulțire**. Se poate observa că proprietatea de mai sus se transferă exponenților din descompunerea în factori primi a fiecărui număr S_i.

Să analizăm același șir $[10, 20, 30] = [2 \cdot 5, 2^2 \cdot 5, 2 \cdot 3 \cdot 5]$ dar aplicând operația de înmulțire. Se obțin succesiv șirurile:

$$[2 \cdot 5, 2^2 \cdot 5, 2 \cdot 3 \cdot 5] \longrightarrow [2^3 \cdot 5^2, 2^3 \cdot 3 \cdot 5^2] \longrightarrow [2^6 \cdot 3 \cdot 5^4]$$

Se poate observa că **exponenții** respectă proprietatea (1):

$$\begin{array}{ccccccc} 6 & = & 1 \cdot C_2^0 & + & 2 \cdot C_2^1 & + & 1 \cdot C_2^2 \\ 1 & = & 0 \cdot C_2^0 & + & 0 \cdot C_2^1 & + & 1 \cdot C_2^2 \\ 4 & = & 1 \cdot C_2^0 & + & 1 \cdot C_2^1 & + & 1 \cdot C_2^2 \end{array}$$

Așadar se poate calcula valoarea TOP din finalul aplicării algoritmului făcând o descompunere a numerelor S_i, $1 \leq i \leq N$ în factori primi și aplicând proprietatea (1). Contribuția (exponentul) unui factor prim la produsul final este dată de suma exponenților săi proveniți din toate elementele vectorului rotit. Produsul final este putere dacă cmmdc-ul exponenților este mai mare decît 1. De exemplu, $2^6 \cdot 3^4 = (2^3 \cdot 3^2)^2 = 72^2$ este putere, pe cînd $2^5 \cdot 3^4$ nu este putere.

Rezolvarea problemei presupune parcurgerea următoarelor etape:

1. **Precalcularea combinărilor**. Se poate face în mai multe moduri însă cel mai eficient este cu triunghiul lui Pascal.
2. **Precalcularea** exponenților din descompunerea în factori primi a fiecărui număr S_i, $1 \leq i \leq N$ în vederea folosirii lor repetate la fiecare permutare circulară.
3. **Determinarea exponenților** numărului TOP pentru fiecare permutare circulară a șirului dat utilizând expresia (1).
4. **Determinarea cmmdc** al exponenților anterior calculați pentru a determina dacă numărul TOP este putere.
5. **Permutarea circulară** către **stânga** a șirului S_i, $1 \leq i \leq N$.

Complexitatea soluției are trei componente:

- Factorizarea celor TN numere. Putem face acest lucru brut, în $O(TN\sqrt{VALMAX})$ sau cu ciurul lui Eratostene în $O(VALMAX \log VALMAX)$. Prima metodă este mai eficientă, mai ales dacă colectăm în prealabil numerele prime pînă la $\sqrt{VALMAX}$.
- Aflarea factorizării produsului final. Aceasta necesită $O(T \cdot N^2 \cdot \log N)$: pentru fiecare test, pentru fiecare rotație și pentru fiecare element, procesăm fiecare divizor al elementului.
- Testul de putere (aflarea cmmdc-ului exponenților). Acesta necesită $O(TN^3 \log N)$: pentru fiecare test, pentru fiecare rotație și pentru cei $O(N \log N)$ factori primi distincți facem o operație cmmdc. Știm că cmmdc-ul face un număr logaritmic de pași, iar valorile pe care operează (exponenții) provin din combinări de N, deci sînt exponențiali în N. De aceea, teoretic trebuie să socotim cmmdc-ul ca având cost $O(N)$.

În practică, dominantă este factorizarea, care merită implementată eficient.

Soluția 2

Iată o soluție diferită, care nu necesită găsirea formulei combinatorice. Calculăm produsele în mod naiv. Întrucît ele vor depăși rapid **`long long`**, factorizăm termenii șirului și îi reprezentăm ca pe liste de perechi (bază, exponent), ordonate după bază. Atunci ca să calculăm produsul a două numere, trebuie să interclasăm cele două liste (factorizări). Cînd întîlnim aceeași bază în ambele factorizări, adunăm exponenții. De exemplu:

$$(2^3 \cdot 7^2 \cdot 11^1) \times (3^1 \cdot 7^4 \cdot 13^2) = (2^3 \cdot 3^1 \cdot 7^6 \cdot 11^1 \cdot 13^2)$$

Astfel, calculăm o matrice A de produse de formă triunghiulară. Pe prima linie așezăm elementele șirului, iar $A_{l,c} = A_{l-1,c} \cdot A_{l-1,c+1}$. Atunci, în $A_{N,1}$ vom obține factorizarea produsului final. Ca și mai sus, calculăm cmmdc-ul exponenților din această factorizare ca să decidem dacă produsul este putere.

Această soluție are complexitatea $O(T \cdot N^4 \cdot \log N)$: pentru fiecare test și fiecare rotație, pentru fiecare din cele $O(N^2)$ produse calculate, interclasează două factorizări care, spre final, vor ajunge la $O(N \log N)$ factori. Soluția va obține circa 56 de puncte.

Putem reduce complexitatea dacă calculăm simultan răspunsul pentru **toate** rotațiile. Pe fiecare linie a matricei calculăm toate celulele, inclusiv ultima celulă, căreia îi dăm valoarea $A_{l,N} = A_{l-1,N} \cdot A_{l-1,1}$. Atunci în cele N celule ale ultimei linii vom obține exact produsele celor N rotații.

Această soluție are complexitatea $O(T \cdot N^3 \cdot \log N)$, egală cu Soluția 1. Ea se comportă bine în practică, de exemplu pentru că este greu de construit un vector care chiar să conțină $O(N \log N)$ factori primi distincți.

Problema 3. Sumgcd

Propusă de: Prof. Mihai Bunget, Colegiul Național Tudor Vladimirescu, Târgu-Jiu

Subtask 1

Pentru a calcula suma $S(A)$ se parcurge șirul A și pentru fiecare element A_i, cu $i \geq 2$, se calculează numărul B_i prin parcurgerea șirului A între indicii 1 și $i-1$ și calcularea celui mai mare divizor corespunzător.

Complexitate $O(N^2)$.

Subtask 2

Se determină divizorii numărului A_1 și se marchează cu 1 existența acestor divizori într-un vector de vizitare. Pentru fiecare termen din șirul A, începând cu al doilea, se determină divizorii acestuia și se reține cel mai mare divizor care a fost vizitat anterior. Acest divizor se adaugă la suma $S(A)$ și se actualizează vectorul de vizitare cu toți divizorii termenului curent.

Determinarea divizorilor lui A_i se poate face fie prin parcurgerea numerelor de la 1 la $\sqrt{A_i}$ și verificarea dacă acestea divid pe A_i, fie prin determinarea factorilor primi din descompunere și generarea divizorilor prin metoda backtracking.

Complexitate $O(N \cdot \sqrt{V})$ sau $O(V \cdot \log V)$ în funcție de metoda de determinare a divizorilor fiecărui termen al șirului A, unde V este valoarea maximă a valorilor termenilor din șirul A.

Subtask 2 - soluție alternativă

Observație: să presupunem că valoarea X apare în A pe pozițiile $i_1, i_2, \ldots, i_k$ (poziții în ordine crescătoare). Atunci $B_{i_2}, B_{i_3}, \ldots, B_{i_k}$ vor fi toate egale cu X. Cu alte cuvinte toate aparițiile valorii X în vector vor avea B_i asociat egal chiar cu X, mai puțin prima apariție.

De aceea vom calcula doar valorile din vectorul B ale primelor apariții ale valorilor din A. Pentru aceasta vom reține într-un vector caracteristic $P[]$ indicele primei apariții ale valorilor din A. Astfel $P[X]$ = indicele primei apariții ale valorii X.

În continuare vom folosi un algoritm tip ciur al lui Eratostene. Pentru fiecare valoare X din ciur vom parcurge valorile $D = X, 2X, 3X, \ldots$, etc. Pentru fiecare valoare D ne întrebăm dacă există printre valorile de la intrare consultând vectorul $P[]$. Vom reține două lucruri: numărul K de valori D care apar în A precum și M, indicele minim al apariției unei valori D.

Dacă K este cel puțin 2 înseamnă că vom avea valori la intrare al căror număr B este cel puțin X și marcăm acest lucru. Mai exact, pentru toate valorile D găsite în A mai puțin cea de la indicele M vom marca B asociat ca fiind X.

La final însumăm valorile B. Complexitate: $O(N + V \log V)$. Iată pseudocodul, considerând vectorul A citit:

Calcul vector caracteristic $P[]$

```
Inițializează P[] la zero
for i ← 1 to N do
    if P[A[i]] = 0 then
        P[A[i]] ← i
    else
        B[i] ← A[i]
    end if
end for
```

Calcul vector $B[]$

```
Inițializează B[] la 1
V ← maximul valorilor din A
for X ← 2 to V do
    K ← 0                                   ▷ numărul de numere din A divizibile cu X
    M ← N + 1                               ▷ indice minim al unui număr divizibil cu X (inițial infinit)
    for D ← X to V cu pas X do
        if P[D] > 0 then
            K ← K + 1
            if P[D] < M then atunci
                M ← P[D]
            end if
        end if
    end for
    if K > 1 then                           ▷ reparcurgem valorile D
        for D ← X to V cu pas X do
            if P[D] > 0 și P[D] > M then
                B[P[D]] ← X
            end if
        end for
    end if
end for
```

Calcul suma B

```
S ← suma valorilor din B
```

Subtask 3

Cum N are o valoare mică se poate folosi metoda backtracking pentru generarea permutărilor șirului A (sau `next_permutation` din STL) și calculul pentru fiecare permutare a numărului $S(A)$.

Complexitate $O(N! \cdot \log V)$.

Subtask 4

Faptul că pentru fiecare termen A_i, cu $i \geq 2$, trebuie să găsim un termen anterior care maximizează cel mai mare divizor comun ne sugerează că fiecare termen „se leagă" de un termen anterior, ceea ce indică existența unei structuri arborescente.

Această intuiție este corectă. Putem reprezenta șirul ca pe un graf complet cu N noduri în care fiecărui termen îi corespunde un nod, iar costul pe muchia (A_u, A_v) este $-\text{cmmdc}(A_u, A_v)$. În acest graf, un arbore parțial de cost minim (APM) va consta dintr-o colecție de muchii care minimizează suma cu minus a cmmdc-urilor, așadar va maximiza suma cmmdc-urilor.

Astfel se poate aplica algoritmul lui Kruskal sau Prim pentru a determina arborele parțial de cost minim.

Implementare cu Algoritmul lui Kruskal

Se determină mai întâi divizorii numerelor din șirul A și pentru fiecare divizor se formează o listă cu indicii termenilor care au acest divizor.

Inițial fiecare termen al șirului A se consideră ca făcând parte dintr-o mulțime cu un element, acesta fiind considerat rădăcina unui arbore. Se parcurg descrescător valorile de la V la 1 (acestea fiind posibile costuri ale muchiilor) și, pentru fiecare valoare, dacă în lista corespunzătoare se află cel puțin două noduri se adaugă o muchie între primul nod din listă și oricare alt nod din listă numai în cazul în care cele două noduri fac parte din mulțimi (subarbori) diferite. De fiecare dată se adaugă la suma $S(A)$ valoarea muchiei și se unesc cele două mulțimi (subarbori) prin subordonarea rădăcinii unui subarbore la rădăcina celuilalt.

În momentul în care au fost adăugate $N - 1$ muchii și corespunzător costul muchiei la suma $S(A)$ atunci se obține valoarea maximă a acestei sume.

Complexitate $O(V \cdot \log V)$.

Implementare cu algoritmul lui Prim

Algoritmul lui Prim este un greedy pe care îl putem intui natural și dacă nu găsim reducerea formală la APM. Inițializăm permutarea cu orice element, apoi de $N-1$ ori adăugăm la permutare acel element care maximizează cmmdc-ul între elementele deja alese (cele din permutare) și cele încă nealese. Aceasta este fix ideea algoritmului lui Prim, care pentru grafuri clasice se implementează cu heap-uri. Numim elementele alese „jumătatea stîngă", iar numerele încă nealese „jumătatea dreaptă".

Pentru problema de față menținem, pentru jumătățile stîngă și dreaptă, două structuri de date similare. Pentru fiecare divizor d, ținem minte un contor: cîte numere din mulțime se divid cu d. Atunci, prin definiție, cmmdc-ul maxim între cele două jumătăți este d-ul maxim care are un contor nenul în ambele jumătăți. Apar, deci, trei nevoi:

1. Să găsim rapid d-ul maxim.
2. Să găsim un număr x din jumătatea dreaptă care se divide cu d.
3. Să-l mutăm pe x în stînga.

Pentru (2), este suficient să ținem, pentru fiecare divizor d, lista elementelor din dreapta care sînt multipli de d. Memoria este $O(N \log N)$, acceptabilă.

Pentru (3), trebuie să iterăm prin divizorii lui x (de exemplu recursiv), să decrementăm contoarele din dreapta și să le incrementăm pe cele din stînga.

Pentru (1), să observăm că în stînga contoarele doar cresc, iar în dreapta doar scad. Atunci putem menține un AIB sau un arbore de intervale care stochează 1 pe pozițiile d unde contoarele sînt pozitive în ambele jumătăți. Cînd mutăm primul multiplu de d din dreapta în stînga, scriem 1 în AIB pe poziția d, iar cînd multăm ultimul multiplu de d, scriem 0 în AIB pe poziția d. Cu această informație, răspunsul la (1) este dat de poziția maximă a unui 1 în AIB, pe care o putem găsi cu o căutare binară în $O(\log MAX\_VAL)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Prof. Adrian Panaete, Colegiul Național „A.T. Laurian” Botoșani
- Prof. Ciprian Chește, Liceul Tehnologic „Grigore C. Moisil” Buzău
- Instr. Cristian Frâncu, Nerdvana București
- Instr. Cătălin Frâncu, Nerdvana București
- Stud. Andrei Boacă, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Prof. Mihai Bunget, Colegiul Național Tudor Vladimirescu, Târgu-Jiu
- Prof. Gheorghe-Eugen Nodea, Centrul Județean de Excelență Gorj, Târgu-Jiu
- Prof. Emanuela Cerchez, Colegiul Național „Emil Racoviță” Iași
- Stud. Alin Răileanu, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Stud. Victor Botnaru, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- Prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin
- Stud. Răzvan Alexandru Rotaru, Facultatea de Informatică, Universitatea „Alexandru Ioan Cuza”, Iași
- Stud. Rareș-Andrei Cotoi, Universitatea Babeș-Bolyai, Cluj, Facultatea de Matematică și Informatică
- Stud. Giulian Buzatu, Facultatea de Matematică-Informatică, Universitatea București
- Prof. Dan Pracsiu, Liceul Teoretic Emil Racoviță, Vaslui
- Prof. Marinel Șerban, Colegiul Național „Emil Racoviță” Iași

- Stud. Petruț-Rareș Gheorghieș, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București
- Stud. Ioan-Cristian Pop, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica București