

Concursul Național de Informatică „Urmașii lui Moisi”, Clasa a IX-a Descrierea soluțiilor

Comisia științifică

June 6, 2026

Problema 1. Teams

Propusă de: stud. Roșu Adrian-Andrei, Universitatea Națională de Știință și Tehnologie Politehnica București

Subtask 1: $N \leq 300$ și $Q = 0$ (9 puncte)

În primul rând, trebuie să aducem funcția $echilibru(i, j)$ la o formă mai simplă. Observăm că în urma procesului de îndoire, elevul i ajunge să facă pereche cu j , elevul $i + 1$ cu $j - 1$ și așa mai departe, până la mijlocul subsecvenței. Putem astfel rescrie funcția noastră în limbaj matematic astfel:

```
1: function echilibru( $i, j$ )
2:    $S \leftarrow 0$ 
3:   for  $k \leftarrow 0$  to  $\lfloor \frac{j-i}{2} \rfloor$  do
4:      $S \leftarrow S + (v[i+k] \oplus v[j-k])$ 
5:   return  $S$ 
6: end function
```

Pentru a calcula valoarea cerută trebuie să iterăm prin fiecare pereche de indici și să calculăm echilibrul subsecvenței respective. Complexitatea pentru acest subtask este $O(N^3)$.

Subtask 2: $N \leq 3000$ și $Q = 0$ (10 puncte)

Deoarece nu mai putem aplica naiv algoritmul de îndoire pentru fiecare subsecvență trebuie să gândim problema în alt mod. În loc de a încerca să calculăm efectiv valorile pentru fiecare interval (i, j) , putem calcula contribuția pe care o are perechea la suma totală. Această contribuție este egală cu $(v[i] \oplus v[j]) \cdot f(i, j)$, unde $f(i, j)$ este egală cu numărul de câte ori elevul i ajunge să fie coleg cu j . Pentru a calcula $f(i, j)$ trebuie să facem o observație:

Observație 1. Pornind de la subsecvența (i, j) , elevii i și j vor mai fi în aceeași echipă doar în subsecvențele $(i-1, j+1), (i-2, j+2), \dots, (i-k, j+k)$.

Așadar, valoarea lui $f(i, j)$ este egală cu $k+1$. Observăm că această valoare este egală cu $\min(\text{dist}[i], \text{dist}[j])$, unde $\text{dist}[i]$ reprezintă distanța minimă față de cele 2 capete ale șirului, această formulă rezultă din faptul că noi ne extindem până atingem cea mai apropiată margine. Prin definiție $\text{dist}[i] = \min(i, N-i+1)$, ea reprezintă o constantă. Dacă punem toate aceste formule cap la cap, obținem:

$$\sum_{i=1}^N \sum_{j=i}^N \text{echilibru}(i, j) = \sum_{i=1}^N \sum_{j=i}^N (v[i] \oplus v[j]) \cdot \min(\text{dist}[i], \text{dist}[j]).$$

Complexitatea pentru acest subtask este $O(N^2)$.

Subtask 3: $v[i] \in \{0, 1\}$ și $Q = 0$ (18 puncte)

Întrucât vectorul nostru este binar, $v[i] \oplus v[j] \neq 0$ este echivalent cu $v[i] \neq v[j]$. Pentru o poziție i fixată trebuie să calculăm câte poziții j cu $j \geq i$ astfel încât cele 2 valori sunt diferite. Ceea ce ne încurcă este funcția \min . Putem să îi analizăm comportamentul pentru a găsi o forma mai simplă. Avem $\min(\text{dist}[i], \text{dist}[j]) = \min(\min(i, N-i+1), \min(j, N-j+1))$. Cum $i \leq j$, obținem și $N-j+1 \leq N-i+1$. Putem rescrie ecuația inițială $\min(\text{dist}[i], \text{dist}[j]) = \min(i, N-j+1)$. Explicitând funcția obținem:

$$\min(i, N-j+1) = \begin{cases} i, & \text{dacă } j \leq N-i+1 \\ N-j+1, & \text{dacă } j > N-i+1 \end{cases}$$

Suma noastră a minimelor pentru un i fixat devine:

$$\sum_{j=i}^N \min(i, N-j+1) = \sum_{j=i}^{\max(i-1, N-i+1)} i + \sum_{j=\max(i, N-i+2)}^N (N-j+1).$$

Așadar suma completă este:

$$\sum_{j=i}^N (v[i] \oplus v[j]) \cdot \min(i, N-j+1) = \sum_{j=i}^{\max(i-1, N-i+1)} i \cdot (v[i] \oplus v[j]) + \sum_{j=\max(i, N-i+2)}^N (N-j+1) \cdot (v[i] \oplus v[j]).$$

Putem lua cele două sume pe rând. Pentru $\sum_{j=i}^{\max(i-1, N-i+1)} i \cdot (v[i] \oplus v[j])$, îl putem scoate pe i în față și obținem $i \cdot \sum_{j=i}^{\max(i-1, N-i+1)} (v[i] \oplus v[j])$. Cum cele două valori sunt binare, ne interesează câte valori $v[j]$ avem în intervalul $[i, \max(i-1, N-i+1)]$ care să fie diferite de $v[i]$ (adică egale cu $1-v[i]$). Putem calcula această cantitate în $O(1)$ folosind sume parțiale de forma $\text{sum_bit_prefix}[\text{bit}][j]$, unde $\text{sum_bit_prefix}[\text{bit}][j] = \sum_{k=1}^j (v[k] == \text{bit})$.

Trecând la a doua sumă, vom proceda similar, doar că aici ne vom folosi de un tablou de sufixe: $\text{sum_distanțe_sufix}[\text{bit}][j] = \sum_{k=j}^N (v[k] == \text{bit}) \cdot (N-k+1)$.

Pentru a reconstrui complet soluția, notăm cu $val = 1 - v[i]$ valoarea complementară pe care o căutăm. Contribuția totală a elementului i se calculează în $O(1)$ cu următoarea formulă:

$$contribution(i) = i \cdot \left(\text{sum_biti_prefix}[val][\max(i-1, N-i+1)] - \text{sum_biti_prefix}[val][i-1] \right) + \text{sum_distanțe_sufix}[val][\max(i, N-i+2)].$$

Răspunsul devine în final:

$$\sum_{i=1}^N contribution(i).$$

Cum noi calculăm în $O(1)$ toate sumele parțiale, complexitatea totală este $O(N)$.

Subtask 4: $Q = 0$ (6 puncte)

Pentru a putea calcula optim suma cerută fără nicio limită, trebuie să reducem problema la subtask-ul anterior cu valori de 0 și 1. Operația de *sau exclusiv pe biți* este independentă la nivelul fiecărui bit. Așadar, putem itera prin toți cei $\log(\text{MAX_VAL})$ biți și să calculăm contribuția fiecărei poziții exclusiv pentru bitul curent. Tot ce trebuie să facem este să înmulțim contribuția fiecărui bit cu 2^{bit} și să însumăm aceste valori pentru a obține rezultatul complet. Complexitatea finală pentru calcularea sumei tuturor echilibrelor devine astfel $O(N \cdot \log(\text{MAX_VAL}))$.

Subtask 5: $N \leq 3000$ și $N \cdot Q \leq 30\,000\,000$ (8 puncte)

Pentru a putea rezolva scenariile trebuie să înțelegem cum se schimbă suma cerută dacă modificăm un singur element. Putem observa destul de repede că noi putem să *scădem* contribuția valorii inițiale și să o *adăugăm* pe cea nouă. Pentru fiecare dintre cele Q scenarii răspunsul va fi:

$$\sum_{i=1}^N \sum_{j=i}^N \text{echilibru}(i, j) - \text{contribution}(\text{poz}, v[\text{poz}]) + \text{contribution}(\text{poz}, val).$$

Pentru a calcula contribuția putem itera prin toate cele $N - 1$ poziții neschimbate și să adunăm XOR-ul valorilor înmulțit cu distanța minimă până la cel mai apropiat capăt. Formal, $\text{contribution}(\text{poz}, val) = \sum_{i=1}^N (v[i] \oplus val) \cdot \min(\text{dist}[i], \text{dist}[\text{poz}])$.

Obținem complexitatea finală de $O(N^2 + Q \cdot N)$ sau $O(N \cdot \log(\text{MAX_VAL}) + Q \cdot N)$ în funcție de modul în care calculăm suma inițială.

Notă: O soluție care schimbă valoarea în memorie apoi rulează algoritmul în $O(N \cdot \log(\text{MAX_VAL}))$ pentru calcularea răspunsului pentru fiecare scenariu nu se va încadra în limitele de timp impuse.

Subtask 6: Atât inițial, cât și după orice scenariu, există cel mult 2 elemente nenule în șir. (13 puncte)

Pentru a calcula suma totală știind că avem cel mult 2 elemente nenule, putem aborda problema calculând contribuții independente. În primul rând, presupunem pentru fiecare elev aflat la

poziția poz că este singurul element nenul din șir. Contribuția sa, interacționând doar cu zerouri, ar fi $\sum_{i=1}^N \min(dist[i], dist[poz]) \cdot v[poz]$, întrucât $v[poz] \oplus 0 = v[poz]$.

Dacă șirul conține un al doilea element nenul, calculul de mai sus procesează interacțiunea dintre cele două poziții (să le notăm poz_1 și poz_2) în mod eronat: o dată ca $v[poz_1] \oplus 0$ (adăugând $v[poz_1]$) și o dată ca $v[poz_2] \oplus 0$ (adăugând $v[poz_2]$).

Pentru a obține rezultatul corect, trebuie să scădem aceste două contribuții false și să o adăugăm pe cea reală. Astfel, corecția aplicată sumei este:

$$\min(dist[poz_1], dist[poz_2]) \cdot ((v[poz_1] \oplus v[poz_2]) - v[poz_1] - v[poz_2])$$

Observăm că suma distanțelor pentru un punct fixat, $S(poz) = \sum_{i=1}^N \min(dist[i], dist[poz])$, este o constantă ce depinde doar de poziție și poate fi precalculată. Astfel, la fiecare scenariu modificăm valoarea în memorie, ținem o evidență a pozițiilor nenule și recalculăm răspunsul în $O(1)$. Complexitatea totală a acestui subtask devine $O(N + Q)$.

Subtask 7: $poz \in \{1, N\}$ (14 puncte)

Pentru ultimele două subtask-uri trebuie să calculăm suma inițială în complexitate optimă. Întrucât poz se află la unul dintre cele două capete ale șirului, $\min(dist[poz], dist[i]) = 1$ pentru orice i . Așadar funcția $contribution(poz, val) = \sum_{i=1}^N (v[i] \oplus val)$. Această sumă poate fi calculată bit cu bit, cu ajutorul sumelor parțiale în $O(\log MAX_VAL)$. Complexitatea finală este $O(N \cdot \log MAX_VAL + Q \cdot \log MAX_VAL)$.

Subtask 8: Fără restricții suplimentare (100 de puncte)

Ne amintim funcția de contribuție pentru o poziție poz și un bit fixat bit : vrem să calculăm suma $\sum_{i=1}^N (v[i]_{bit} \oplus val_{bit}) \cdot \min(dist[i], dist[poz])$. Fie $D = dist[poz]$. Pentru a scăpa de funcția \min , o putem explicita în funcție de poziția curentă i :

$$\min(dist[i], D) = \begin{cases} i, & \text{dacă } i \leq D \\ D, & \text{dacă } D < i < N - D + 1 \\ N - i + 1, & \text{dacă } i \geq N - D + 1 \end{cases}$$

Această descompunere pe trei ramuri împarte practic șirul într-un prefix, o zonă de mijloc și un sufix. Pentru a calcula suma în $O(1)$, precalculăm pentru fiecare bit $bit \in [0, 30]$ și fiecare valoare binară $b \in \{0, 1\}$ următoarele trei tablouri de sume parțiale:

- $sum_bit_i_prefix[bit][b][i] = \text{numărul de elemente din prefixul } 1 \dots i \text{ care au bitul } bit \text{ egal cu } b$;
- $sum_distanțe_prefix[bit][b][i] = \sum_{k=1}^i k \cdot [v[k]_{bit} == b]$ (corespunzător primei ramuri);
- $sum_distanțe_sufix[bit][b][i] = \sum_{k=i}^N (N - k + 1) \cdot [v[k]_{bit} == b]$ (corespunzător celei de-a treia ramuri).

Observăm că primul și ultimul tablou reprezintă de fapt o extindere a celor din subtask-ul 3. În practică dacă la subtask-ul 3 evaluăm contribuția poziției i cu toate pozițiile j cu $i \leq j$, acum evaluăm cu toate cele N valori, de unde obținem încă o ramură.

Pentru un scenariu (poz, val) , iterăm prin toți biții. Pentru bitul curent bit , căutăm să interacționăm cu biții opuși, deci ne interesează valoarea binară $b = 1 - val_{bit}$. Contribuția adusă pe acest bit se obține apelând funcția $contribution(poz, bit)$, calculată prin adunarea contribuțiilor celor 3 ramuri:

$$\begin{aligned} contribution(poz, bit) = & \left(\text{sum_distant_prefix}[bit][b][D] \right) \\ & + D \cdot \left(\text{sum_bit_prefix}[bit][b][N - D] - \text{sum_bit_prefix}[bit][b][D] \right) \\ & + \left(\text{sum_distant_sufix}[bit][b][N - D + 1] \right) \end{aligned}$$

Înmulțind $contribution(poz, bit)$ cu 2^{bit} pentru fiecare bit și adunând rezultatele, evaluăm funcția totală de contribuție în doar $O(\log(\text{MAX_VAL}))$ operații. Răspunsul pentru fiecare scenariu se calculează scăzând din totalul inițial contribuția valorii vechi și adăugând-o pe cea a valorii noi.

Complexitatea soluției finale este $O(N \cdot \log(\text{MAX_VAL}) + Q \cdot \log(\text{MAX_VAL}))$ și obține 100 de puncte.

Problema 2. Perechi

*Propusă de: prof. Șerban Marin, Centrul Județean de Excelență Iași
stud. Răileanu Alin-Gabriel, Universitatea "Alexandru Ioan Cuza" Iași*

Se cere, pentru fiecare dintre cele T interogări, al n -lea termen al șirului al treilea: din șirul numerelor prime se concatenează fiecare prim de pe poziție impară cu următorul, iar dintre termenii obținuți se păstrează doar cei care sunt, la rândul lor, primi.

Observație 2. Pentru o pereche (a, b) de prime consecutive, $\text{concat}(a, b) \equiv a + b \pmod{3}$ (suma cifrelor, fiindcă $10 \equiv 1 \pmod{3}$). Cum termenul este > 3 , el este compus ori de câte ori $a + b \equiv 0 \pmod{3}$. Așadar, **doar** perechile cu $a + b \not\equiv 0 \pmod{3}$ pot produce un termen al șirului. Empiric, condiția elimină $\approx 57\%$ dintre perechi înainte de orice concatenare a termenilor sau test de primalitate.

Observație 3. Cel mai mare termen are 14 cifre ($\approx 5.6 \times 10^{13}$). Pentru un test de primalitate prin împărțiri sunt necesari divizori până la $\sqrt{5.6 \times 10^{13}} \approx 7.5 \times 10^6$ (strict mai mult decât cel mai mare prim folosit efectiv într-o pereche ($\approx 5.6 \times 10^6$)). Ciurul se dimensionează după această limită, nu după cel mai mare prim.

Soluții parțiale

Toate soluțiile pleacă de la același schelet:

```

void sieve(int LIMIT);           // genereaza lista de prime
long long concat(long long a, long long b); // a * 10^(cifre b) + b
bool isPrime(long long x);       // test de primalitate
long long getTerm(int n);        // al n-lea termen al sirului 3

```

sieve. Ciur al lui Eratostene până la $\approx 8 \times 10^6$ (Observația 3). Pentru subtask-urile cele mai mici se poate renunța la ciur, generând primele prin verificare individuală.

isPrime. Pentru optimizare, se pot verifica ca și divizori direct numerele prime generate de ciur.

getTerm. Se parcurg perechile de prime consecutive de la început; pentru fiecare se aplică filtrul din Observația 2, se formează *concat* și se testează cu *isPrime*, numărând termenii până la al *n*-lea. Termenii deja calculați se memoizează, astfel încât interogările grupate sau repetate dintr-un test nu refac munca.

În funcție de calitatea ciurului, a testului de primalitate și de memoizare, această soluție obține tipic subtask-urile 1–3 ($N \leq 1500$), eventual și subtask-ul 4 ($N \leq 3000$) dacă implementarea este optimizată mai puternic.

Soluții eficiente

Scheletul rămâne identic; se schimbă doar *isPrime* și *getTerm*, adică modul în care ocolim costul testului de primalitate. Există două variante.

Varianta 1: Hardcoding

Ce putem face când avem un șir fix, problema cere să computăm anumiți termeni ai șirului, avem o soluție care primește TLE, timp suficient rămas din concurs și limita suficient de mare pentru dimensiunea maximă a submisiei? Exact, hardcoding.

Împărțim cei 10 000 de termeni în grupe de câte 100 și, calculându-i o singură dată offline, reținem în sursă, pentru fiecare graniță de grupă (100, 200, ...), poziția în lista de prime a primului prim al perechii care a produs acel termen. 100 de valori $\rightarrow \approx 2.3$ KB.

getTerm sare direct la checkpoint-ul grupei $\lfloor n/100 \rfloor$ și reconstruiește acea grupă (cel mult 100 de termeni), memoizând rezultatul; *isPrime* rămâne neschimbat.

Observație 4. Restricția „diferența dintre cea mai mare și cea mai mică valoare *n* dintr-un test este cel mult 100” garantează că toate interogările unui test cad în cel mult două grupe vecine.

Sortând interogările descrescător (cea mai mare dintr-o grupă le adaugă în memoizare pe toate celelalte), soluția face cel mult două reconstrucții de grupă pe test.

Această soluție obține punctajul maxim.

Varianta 2: Optimizarea testului de primalitate

O altă variantă de optimizare este folosirea testului Miller-Rabin, evitând hardcodarea. Această soluție obține, de asemenea, punctajul maxim.

Echipa

Problemele pentru acest concurs au fost pregătite de: