

Olimpiada Națională de Informatică,

Etapa națională, Clasa a IX-a

Descrierea soluțiilor

Comisia științifică

March 27, 2026

Problema 1. SCUDERIA

Propusă de: Student *Mircea Măierean*, ETH, Zurich

Noțiuni semantice

1. Operator $\lceil n \rceil$ reprezintă aproximarea prin adaos la unități a numărului n :
Exemple: $\lceil 2.1 \rceil = 3$, $\lceil 2 \rceil = 2$
2. a este șirul cu n elemente din enunț, indexat de la 0.
3. Prin noțiunea de **consecutiv circular** în cadrul unui șir v de lungime l , indexat de la 0, înțelegem că succesorul elementului de pe poziția $l - 1$ este elementul de pe poziția 0. Astfel, orice indice i va fi tratat prin prisma valorii $i \% l$.
Exemplu: Pentru un șir s de lungime $k = 7$, elementele consecutive circular începând de la poziția 6 pentru o fereastră de lungime $p = 3$ sunt $s[6]$, $s[(6 + 1) \% 7]$ și $s[(6 + 2) \% 7]$, adică $s[6]$, $s[0]$ și $s[1]$.

Observații esențiale

1. Faptul că un indice j aparține mulțimii dacă și numai dacă indicele $j + k$ aparține aceleiași mulțimi, **ori de câte ori ambii indici sunt cuprinși între 0 și $n - 1$** , este echivalent cu: $j - k$ aparține mulțimii dacă și numai dacă indicele j aparține aceleiași mulțimi, **ori de câte ori ambii indici sunt cuprinși între 0 și $n - 1$** .
2. $j \% k = (j + k) \% k$

Subtask 1: $k = 1$ (10 puncte)

Singura valoare validă a lui r este 0. Dacă $p > 0$, Luis va selecta toate elementele din șir, și răspunsul va fi suma elementelor din șir. Dacă $p = 0$, $q > 0$, atunci răspunsul va fi valoarea maximă a sumei a q numere consecutive. Dacă $p = 0$, $q = 0$, atunci răspunsul va fi 0.

Subtask 2: $p = 0, n \leq 1\,000$ (8 puncte)

Dacă $p = 0$, doar Carol va selecta numere. Problema se transformă în a determina subsecvența maximă a $q \cdot k$ numere, iar indicele de început al subsecvenței să aibă restul împărții la $k = r$. Pentru fiecare indice valid, calculăm subsecvența într-un mod iterativ. Complexitatea finală e $O(q \cdot k \cdot \lceil \frac{n}{k} \rceil)$.

Subtask 3: $p = 0$ (15 puncte)

Soluția este extrem de similară cu cea a subtaskului 2. Însă, în loc să calculăm iterativ pentru fiecare valoare, vor reține un șir s de sume parțiale.

$s[i]$ -suma elementelor aflate între pozițiile 0 și i

Definim relația de recurență pentru șirul sumelor parțiale s astfel:

$$s[i] = \begin{cases} a[0] & \text{dacă } i = 0 \\ s[i-1] + a[i] & \text{dacă } i > 0 \end{cases}$$

Această construcție ne permite să calculăm suma elementelor din intervalul $[i, j]$ în timp constant, folosind formula:

$$\text{Suma}(i, j) = s[j] - s[i-1]$$

Pentru fiecare indice $i, i \% k = r$, indicele de final al subsecvenței va fi $i + q \cdot k - 1$. Dacă $i + q \cdot k - 1 < n$, atunci avem o subsecvență validă, a cărei sumă o vom calcula folosind metoda mai sus ($s[i + q \cdot k - 1] - s[i-1]$). Complexitatea finală e $O(n)$, putând fi îmbunătățită la $O(\lceil \frac{n}{k} \rceil)$, dacă iterația pentru verificare poziției începe din start de la r și are pasul k , folosind, spre exemplu: `for (int i = r; i + q * k - 1 < n; i += k).`

Subtask 4: $q = 0, p = 1$ (13 puncte)

Pentru acest subtask, și pe baza observației de mai sus, trebuie să calculăm suma maximă a numerelor care se află la distanță k unul de celălalt. Pentru fiecare din primele k numere, mai facem o iterație având pasul k , și calculăm suma acestor numere. Dintre toate aceste numere, selectăm suma maximă. Complexitatea finală e $O(n)$.

Subtask 5: $q = 0, k \leq 1\,000$ (7 puncte)

Soluția extinde logica subtaskului precedent prin gruparea elementelor în funcție de restul indicilor la împărțirea cu k . Definim șirul sr de dimensiune k , unde fiecare element $sr[r], 0 \leq r < k$ reprezintă suma tuturor valorilor din șirul original care au indicii i , iar $i \% k = r$.

$$sr[r] = \sum_{i \% k = r} a[i], \quad \forall r \in \{0, 1, \dots, k-1\}$$

Conform observației făcute la început, problema se reduce la identificarea unei subsecvențe circulare de lungime p în șirul s care să maximizeze suma totală.

Foarte important: Datorită modului în care Luis face selecția, elementele alese pot proveni din resturi consecutive care „trec” de capătul șirului s . Spre exemplu, pentru $n = 10, k = 4$ și $p = 2$, putem selecta inițial valorile de pe pozițiile 3 și 4, corespunzătoare resturilor 3 și 0, iar la final, mulțimea de indici rezultați să fie $\{0, 3, 4, 7, 8\}$.

Așadar, trebuie de asemenea efectuată o verificare a consecutivității circulare a sumei pe șirul sr . Matematic, suma pentru o subsecvență de lungime p care începe la indexul i se definește prin relația de recurență circulară:

$$\text{Suma}(i, p) = \sum_{j=0}^{p-1} sr[(i + j) \bmod k]$$

Problema se reduce astfel la determinarea maximului acestei sume pentru $i \in \{0, 1, \dots, k-1\}$, procedeu similar cu cel descris la subtaskul 2, dar adaptat pentru structura circulară a resturilor. Complexitatea finală e $O(k \cdot p)$.

Subtask 6: $q = 0$ (12 puncte)

Repetăm procedeu de la subtaskul 5, însă reținem treptat suma a p valori consecutive din șirul de sume parțiale. După, iterând cu un indice i de la p până la $k + p - 1$ inclusiv, adăugăm elementul aflat la $sr[i\%k]$ și scădem elementul aflat la poziția $sr[(i - p)\%k]$. Dintre toate aceste valori, o reținem pe cea cu valoarea maximă. Complexitatea finală e $O(k + p)$.

Subtask 7: $q = 1, p = 1$ (7 puncte)

Fiind ambele valori 1, cu siguranță va exista **un singur element** care se află atât în mulțime lui Carol, cât și în mulțimea lui Luis. Vom combina ideile de la subtaskurile anterioare.

Începem prin a calcula sumele parțiale pentru pozițiile corespunzătoare resturilor, asemenea subtaskului 5. După aceea, similar cu subtaskul 3, vom parcurge toate subsecvențele din k în k cu un indice i , începând de la r până la $n - k - 1$. Vom calcula suma pe subsecvența $i, i + 1, \dots, i + k - 1$, iar, pentru fiecare poziție j , $i \leq j \leq i + k$, vom adăuga $sr[j\%k]$, și vom scădea elementul aflat în ambele mulțimi, adică $a[j]$.

La final, reținem valoarea maximă care se poate obține astfel. Complexitatea finală e $O(n)$.

Ultimele subtaskuri

Pentru ultimele subtaskuri, trebuie să schimbăm perspectiva asupra șirului. Modul de construcție al celor două mulțimi ne sugerează să privim șirul ca fiind o matrice în care sunt în total n elemente.

Matricea are k coloane și $\lceil \frac{n}{k} \rceil$ linii. Chiar dacă $n \% k \neq 0$, putem adăuga pe ultimul rând valori de 0, doar trebuie să avem grijă când selectăm mulțimea lui Carol (vezi 2).

De asemenea, în cazul în care $r \neq 0$, putem să adăugăm $k - r$ elemente cu valoare 0 la începutul șirului. Astfel, în șirul virtual construit, condiția de selecție a lui Carol se transformă:

$$j \% k = r \implies (j + k - r) \% k = (r + k - r) \% k \iff (j + k - r) \% k = k \% k = 0$$

Condiția devine astfel echivalentă cazului în care $r = 0$. Mulțimea lui Luis nu este afectată de aceste artificii, suma ei fiind exact aceeași (0 nu adaugă nimic în plus sumei finale). Echivalent, suma lui Carol devine suma a q linii consecutive din matrice, iar suma lui Luis devine suma a p coloane consecutive circular din matrice. Fiecare indice i din șirul inițial corespunde perechii de indici $\{\frac{i}{k}, i \% k\}$ în matrice.

Pentru primul caz, reprezentat în **Figura 1**, dacă adunăm suma elementelor din dreptunghiul roșu cu suma elementelor aflate în dreptunghiul galben, vom avea suma elementelor aflate în dreptunghiul portocaliu de două ori. Pentru a afla răspunsul final, vom scădea suma elementelor aflate în dreptunghiul portocaliu o dată, având astfel suma elementelor aflate în reuniunea finală a celor doi piloți.

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47

Carol: Liniile 2, 3

Luis: Coloanele 4, 5, 6

Figure 1: Carol alege liniile consecutive $\{2, 3\}$, colorate în roșu. Luis alege coloanele consecutive $\{4, 5, 6\}$, colorate în galben. Zona portocalie reprezintă intersecția celor două selecții. Toate pătratele colorate vor face parte din selecția finală.

Pentru al doilea caz, reprezentat în **Figura 2**, o modalitate de calcul e să calculăm suma tuturor elementelor, din care să scădem suma celor două dreptunghiuri albe. Astfel, obținem suma tuturor elementelor aflate în reuniune.

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47
6	48	49	50	51				

Carol: Liniile 4, 5 **Luis: Coloanele 6, 7, 0**

Figure 2: Carol alege liniile consecutive $\{4, 5\}$, colorate în roșu. Luis alege coloanele consecutive **circular** $\{6, 7, 0\}$, colorate în galben. Zona portocalie reprezintă intersecția celor două selecții. Toate pătratele colorate vor face parte din selecția finală. **Linia 6 nu poate fi selectată de către Carol**, ea fiind incompletă

Diferența dintre subtaskuri constă în eficiența calculării sumelor din dreptunghiuri. Pentru restricțiile în care $n \leq 100$, putem adopta o abordare directă: pentru fiecare configurație validă, iterăm prin toate elementele dreptunghiului delimitat de colțul stânga-sus (i_1, j_1) și cel dreapta-jos (i_2, j_2) .

Fiecărei perechi de indici (i, j) din matrice îi corespunde elementul $a[i \cdot k + j]$ din șirul original. Astfel, suma elementelor din dreptunghi se calculează în timp liniar față de aria acestuia, rezultând o complexitate finală de $O(n^2)$.

Pentru subtaskurile rămase, unde restricțiile devin mai stricte, abordarea iterativă nu mai este eficientă. Optimizarea constă în utilizarea **sumelor parțiale pe matrice**, tehnică ce ne permite să calculăm suma oricărui dreptunghi în timp constant, $O(1)$, după o preprocesare de $O(n)$.

Deoarece dimensiunile matricii depind de valorile citite în timpul rulării, nu putem declara o matrice statică cu dimensiuni fixe. În schimb, vom simula structura bidimensională folosind un vector liniarizat de dimensiune n .

Astfel, orice element aflat la coordonatele (i, j) în matricea conceptuală va fi accesat în șirul nostru prin formula de indexare $i \cdot k + j$. Această metodă ne permite să aplicăm algoritmul de sume parțiale descris anterior fără a consuma memorie suplimentară inutilă și fără a depinde de dimensiuni predefinite la compilare. Pentru ușurință în citire, ne vom referi la acest vector ca

fiind o matrice în rândurile următoare. Prin $a(i, j)$ ne referim la $a[i \cdot k + j]$.

Definim matricea sumelor parțiale S , unde $S[i][j]$ reprezintă suma tuturor elementelor din dreptunghiul cu colțul stânga-sus la $(0, 0)$ și colțul dreapta-jos la (i, j) :

$$S[i][j] = \sum_{r=0}^i \sum_{c=0}^j a(r, c)$$

Relația de recurență pentru construirea eficientă a acestei matrici, este:

$$S[i][j] = a(i, j) + S[i-1][j] + S[i][j-1] - S[i-1][j-1]$$

cu cazurile de bază tratate corespunzător pentru indicii egali cu 0. Odată construită această matrice, suma elementelor din orice dreptunghi delimitat de (i_1, j_1) și (i_2, j_2) se obține instantaneu prin formula:

$$\text{Sumă}((i_1, j_1), (i_2, j_2)) = S[i_2][j_2] - S[i_1-1][j_2] - S[i_2][j_1-1] + S[i_1-1][j_1-1]$$

Astfel, complexitatea finală a algoritmului devine $O(n)$. Algoritmul final integrează toate sub-taskurile, restricțiile neinfluențând modul de calcul al matricii și al sumelor din dreptunghiuri.

Observații extra soluție: Există metode de a alocă matricea în timpul rulării, însă acestea nu vor obține 100 de puncte, depășind limita de memorie impusă problemei. Acest lucru se poate face fie utilizând librăria `vector`, fie utilizând operatorul `new`.

Problema 2. POLEN

Propusă de: Student **Raul Ardelean**, Universitatea "Babeș-Bolyai",
Facultatea de Matematică și Informatică, Cluj-Napoca

Cerința 1

În această cerință, înfloririle sunt grupate în perechi

$$(1, 2), (3, 4), \dots, (Q-1, Q).$$

Potrivit enunțului, cele două înfloriri din aceeași pereche au aceeași valoare val , semnale opuse și sunt amplasate astfel încât zona produsă de a doua să completeze exact zona produsă de prima.

Observația-cheie. Pentru fiecare pereche, reuniunea celor două triunghiuri este exact un pătrat, de latură $K+1$, unde K este puterea primei înfloriri din pereche. Mai precis, primul triunghi pornește dintr-un colț; al doilea triunghi, cu puterea $K-1$, pornește din colțul opus; cele două zone acoperă împreună exact pătratul, iar valorile val fiind egale, fiecare pereche este echivalentă cu un singur update de tip „adaugă val pe un pătrat”. Prin urmare, Cerința 1 se reduce la o problemă clasică de actualizări pe pătrate.

Subtask 1 ($N, M, Q \leq 200$). Pentru fiecare pereche, determinăm colțurile pătratului corespunzător, parcurgem toate celulele sale și adăugăm valoarea val . Dacă latura pătratului este $K + 1$, atunci complexitatea unei astfel de actualizări este $O(K^2)$.

Complexitate:

$$O(Q \cdot K^2 + N \cdot M).$$

Subtask 2 ($Q \leq 5000$). Nu mai actualizăm individual fiecare celulă a pătratului. Pentru fiecare linie din pătrat, actualizăm doar intervalul de coloane acoperit. Dacă pe linia i trebuie adăugată valoarea val pe intervalul $[L, R]$, folosim un vector de diferențe pe acea linie, tehnică cunoscută și sub numele de „Șmenul lui Mars”:

$$sp[i][L] += val, \quad sp[i][R + 1] -= val.$$

După ce procesăm toate perechile, reconstruim fiecare linie prin sumă prefix. Astfel, un pătrat de latură $K + 1$ se procesează în $O(K)$ în loc de $O(K^2)$.

Complexitate:

$$O(Q \cdot K + N \cdot M).$$

Subtask 3. Acum dorim să efectuăm fiecare update pe pătrat în $O(1)$. Pentru aceasta, folosim o matrice de diferențe, adică extensia bidimensională a tehnicii cunoscute sub numele de „Șmenul lui Mars”. Dacă pătratul are colțurile (r_1, c_1) , (r_2, c_2) , atunci aplicăm următoarele actualizări:

$$diff[r_1][c_1] += val,$$

$$diff[r_1][c_2 + 1] -= val,$$

$$diff[r_2 + 1][c_1] -= val,$$

$$diff[r_2 + 1][c_2 + 1] += val.$$

La final reconstruim matricea prin sume parțiale 2D.

Complexitate:

$$O(Q + N \cdot M).$$

Cerința 2

În această cerință, pentru fiecare înflorire, toate cele patru semnale sunt active simultan.

Observația geometrică. Reuniunea geometrică a celor patru triunghiuri formează un romb de rază K , centrat în (r, c) . Cu toate acestea, nu putem trata operația ca pe o simplă adăugare a valorii val pe întregul romb, deoarece între triunghiuri apar suprapuneri.

Multiplicități. Pentru o înflorire cu centrul în (r, c) și valoarea val :

- orice celulă din romb este acoperită de cel puțin un triunghi, deci primește contribuția val ;
- orice celulă aflată pe linia r , diferită de centru, este acoperită de două triunghiuri, deci primește contribuția $2 \cdot val$;
- orice celulă aflată pe coloana c , diferită de centru, este acoperită de două triunghiuri, deci primește contribuția $2 \cdot val$;
- centrul (r, c) este acoperit de toate cele patru triunghiuri, deci primește contribuția $4 \cdot val$.

Prin urmare, contribuția unei înfloriri poate fi văzută ca suma a patru componente:

1. $+val$ pe întregul romb;
2. încă $+val$ pe linia orizontală prin centru;
3. încă $+val$ pe coloana verticală prin centru;
4. încă $+val$ în centru.

Această descompunere este cheia pentru soluția optimă.

Subtask 4 ($N, M, Q \leq 200$). Aplicăm direct brute force. Pentru fiecare înflorire parcurgem toate celulele din jurul centrului, verificăm dacă se află în romb de rază K și adăugăm contribuția corespunzătoare. În practică, putem itera pe liniile din intervalul $[r - K, r + K]$, iar pe fiecare linie să determinăm intervalul de coloane afectat.

Complexitate:

$$O(Q \cdot K^2 + N \cdot M).$$

Subtask 5 ($Q \leq 5000$). În loc să actualizăm fiecare celulă, lucrăm linie cu linie. Pe o linie aflată la distanța $d = |i - r|$ de centru, rombul afectează intervalul

$$[c - (K - d), c + (K - d)].$$

Putem face un update de interval în $O(1)$ pe acea linie, folosind un vector de diferențe pe linii. În mod echivalent, putem privi rombul ca reuniunea a patru triunghiuri și actualiza separat fiecare contribuție, tot linie cu linie. În ambele variante, pentru o înflorire costul devine $O(K)$.

Complexitate:

$$O(Q \cdot K + N \cdot M).$$

Subtask 6. Pentru a face update-ul în $O(1)$, folosim schimbarea de coordonate $u = i + j, v = i - j$. În aceste coordonate, rombul devine un pătrat. Dacă centrul este (r, c) , atunci punctul corespunzător este $u_0 = r + c, v_0 = r - c$. Rombul de rază K devine pătratul:

$$[u_0 - K, u_0 + K] \times [v_0 - K, v_0 + K].$$

Deci putem face un update de pătrat în $O(1)$ într-o matrice de diferențe în coordonatele (u, v) .

Observație de implementare. Coordonata $v = i - j$ poate fi negativă, deci în implementare o translatăm cu o constantă, de exemplu $v' = v + M$. Astfel, toți indicii rămân pozitivi.

Corectarea suprapunerilor. Update-ul pe dreptunghi ne dă $+val$ pe tot rombul, adică exact contribuția de bază. Mai rămâne să adăugăm corecțiile pentru suprapuneri:

- $+val$ pe linia r , pe intervalul $[c - K, c + K]$;
- $+val$ pe coloana c , pe intervalul $[r - K, r + K]$;
- $+val$ în celula (r, c) .

După toate aceste adăugări obținem:

- val în interiorul rombului;
- $2 \cdot val$ pe crucea centrală;
- $4 \cdot val$ în centru.

Complexitate:

$$O(Q + N \cdot M).$$

Cerința 3

În această cerință poate fi activă orice submulțime din cele patru semnale. Asta înseamnă că fiecare înflorire este suma contribuțiilor unor triunghiuri orientate în una sau mai multe direcții. Nu mai avem nici structura fixă de pătrat din Cerința 1, nici simetria completă a rombului din Cerința 2, deci trebuie să tratăm direct triunghiurile.

Subtask 7 ($N, M, Q \leq 200$). Pentru fiecare înflorire verificăm care semnale sunt active. Pentru fiecare triunghi activ parcurgem toate celulele din triunghi și adăugăm val .

Complexitate:

$$O(Q \cdot K^2 + N \cdot M).$$

Subtask 8 ($Q \leq 5000$). Optimizăm similar cazurilor anterioare: în loc să actualizăm individual fiecare celulă, efectuăm update-uri pe intervale, linie cu linie. Pentru fiecare triunghi activ, determinăm liniile pe care îl intersectează, iar pentru fiecare astfel de linie stabilim intervalul de coloane afectat și aplicăm actualizarea cu ajutorul unui vector de diferențe pe linii. Astfel, fiecare triunghi activ poate fi procesat în $O(K)$.

Complexitate:

$$O(Q \cdot K + N \cdot M).$$

Subtask 9. Acum vrem update în $O(1)$ pentru fiecare triunghi activ. Este suficient să explicăm cazul unui singur tip de triunghi, de exemplu Nord-Est. Celelalte trei orientări se obțin simetric.

Triunghiul Nord-Est. Considerăm triunghiul cu vârfurile (r, c) , $(r - K, c)$, $(r, c + K)$. Acest triunghi afectează liniile $i \in [r - K, r]$. Pe fiecare linie i , intervalul de coloane afectat este $[c, c + i - (r - K)]$.

Observăm două lucruri:

- marginea stângă este fixă: intervalul începe mereu în coloana c ;
- marginea dreaptă se deplasează cu $+1$ la fiecare linie, deci urmează o diagonală.

Așadar, forma triunghiului este determinată de o pornire verticală și o oprire pe diagonală.

Ideea de update. Folosim două matrici de diferențe, una pentru contribuții care pornesc pe verticală și una pentru contribuții care se opresc pe diagonală.

Pasul 1: pornim contribuția.

Punem $+val$ pe coloana c , pentru toate liniile din intervalul $[r - K, r]$. Asta înseamnă că, dacă ulterior facem propagarea corespunzătoare pe fiecare linie spre dreapta, contribuția ar rămâne activă începând din coloana c .

Pasul 2: oprim contribuția după muchia oblică.

Muchia oblică dreaptă a triunghiului este descrisă de punctele $(r - K, c)$, $(r - K + 1, c + 1)$, $\dots$, $(r, c + K)$. Ca să păstrăm doar interiorul triunghiului, trebuie să anulăm contribuția imediat după această muchie. De aceea plasăm $-val$ pe diagonala aflată imediat în dreapta triunghiului. După ce calculăm sumele parțiale potrivite:

- contribuția pornește din coloana c ;
- rămâne activă până la muchia oblică;
- se anulează imediat după ea.

În final, pe fiecare linie rămâne activ exact intervalul $[c, c + i - (r - K)]$, adică exact triunghiul NE.

De ce este $O(1)$? Fiecare triunghi este tratat printr-un număr constant de actualizări: una sau două în structura verticală și una sau două în structura diagonală. Deoarece reconstrucția finală se face o singură dată, prin sume parțiale, rezultă că fiecare triunghi activ are cost $O(1)$.

Celelalte orientări. Celelalte trei tipuri de triunghi se tratează analog:

- **Sud Vest** este simetric lui **Nord Est**;
- **Nord Vest** și **Sud Est** folosesc aceeași idee, dar cu diagonala opusă.

Cu alte cuvinte, există două familii:

- triunghiuri controlate de coloană + diagonală principală;
- triunghiuri controlate de coloană + diagonală secundară.

Prin simetrie, toate cele patru cazuri se reduc la aceeași idee geometrică: pornim contribuția pe o margine dreaptă și o oprim pe muchia oblică.

Complexitate:

$$O(Q + N \cdot M).$$

Subtask 10. În acest subtask, aceeași idee rămâne validă, dar trebuie să tratăm cu atenție porțiunile care ies din matrice. În subtaskurile anterioare puteam presupune că fiecare triunghi se află complet în interiorul matricei. Acum nu mai este adevărat.

Ce se schimbă? Intersecția unui triunghi cu o linie a matricei nu mai este întotdeauna „intervalul ideal” descris de formula geometrică. Ea devine acel interval restrâns la $[1, M]$. La fel, liniile afectate trebuie restrânse la $[1, N]$.

Împărțirea contribuției. Pentru fiecare triunghi putem separa contribuția în două zone:

1. **zona în care muchia oblică este încă în interiorul matricei.** Aici folosim exact aceeași combinație de update vertical + update pe diagonală ca la subtaskul 9.
2. **zona în care triunghiul a depășit marginile matricei.** Aici, pe liniile afectate, intersecția cu matricea devine un prefix complet sau un sufix complet al liniei. Această parte poate fi tratată cu update-uri obișnuite de interval.

Prin urmare, nu schimbăm ideea de bază. Doar restrângem contribuția la porțiunea aflată în interiorul matricei, iar partea rămasă o descompunem într-un număr constant de update-uri. Și în acest caz fiecare triunghi activ poate fi procesat în $O(1)$.

Complexitate:

$$O(Q + N \cdot M).$$

Problema 3. CANONIC

*Propusă de: Student Andrei-Cristian Ivan, Universitatea Națională de Științe și Tehnologie Politehnica,
Facultatea de Automatică și Calculatoare, București*

Cerința 2, $V_i = 1$ (2 puncte)

Cum toate elementele șirului se divid unul la celălalt, rezultă că toate subsecvențe șirului sunt subsecvențe canonice. Complexitate finală $O(1)$.

Cerința 1 + 2, $N \leq 500$ (7 + 6 puncte)

Pentru acest subtask, se pot fixa capetele tuturor subsecvențelor și se poate verifica iterativ dacă subsecvența este canonică sau nu. Complexitate $O(N^3)$.

Cerința 1 + 2, $N \leq 2\,000$ (8 + 13 puncte)

Pentru acest subtask, mai întâi, vom fixa capătul stânga i al subsecvenței, și apoi, vom începe să fixăm capătul dreapta j ($j \geq i$). Dacă V_j nu se divide cu V_i , atunci, este evident că pentru orice $j' \geq j$, subsecvența $V_i, \dots, V_{j'}$ nu va fi canonică. Așadar, pentru un capăt stânga fixat, putem obține toate subsecvențele canonice în timp liniar. Complexitate finală $O(N^2)$.

Cerința 1 + 2, $N \leq 100\,000$, $V_i \leq 1\,000$ (10 + 17 puncte)

Înainte de a rezolva acest subtask, o să facem următoarea observație: condiția ca subsecvența $V_i, V_{i+1}, \dots, V_j$ să fie canonică se poate rescrie ca V_i să fie cel mai mare divizor comun al subsecvenței (1).

Folosindu-ne de observația de mai sus și ținând cont că valorile sunt până în 1 000, putem să facem descompunerea în factori primi a fiecărui număr. Pentru determinarea numerelor prime putem folosi Ciurul lui Eratostene. Vom reține următorul tablou $dr[i][j][k]$ = cea mai din dreapta poziție mai mare sau egală cu i , astfel încât toate numerele din subsecvența $V_i, \dots, V_{dr[i][j][k]}$ conțin factorul prim j cel puțin la puterea k . Vom parcurge șirul de la dreapta spre stânga și vom actualiza tabloul dr la fiecare pas. Fie factorizarea în factori primi a lui $V_i = x_1^{y_1} \cdot x_2^{y_2} \cdot \dots \cdot x_t^{y_t}$, $y_p \geq 1$ pentru orice $1 \leq p \leq t$, unde x_p este un factor prim, iar y_p , puterea la care apare factorul x_p . Atunci, lungimea maximă a unei subsecvențe canonice care are capătul stâng fixat în poziția i se va afla în $dr[i][x_p][y_p]$ minim. Cum tabloul dr de la poziția i actualizează informații din tabloul dr de la poziția $i + 1$, putem renunța la prima dimensiune. Complexitate finală $O(N \cdot |P|)$ cu o constantă dată de puterea maximă la care se găsește fiecare număr prim, iar $|P|$ = cardinalul mulțimii numerelor prime $\leq 1\,000$.

Soluția oficială

Fie tabloul $gcdsecv[i][j]$ = cel mai mare divizor comun al subsecvenței $V_i, \dots, V_j$. Datorită proprietăților celui mai mare divizor comun, $gcdsecv[i][j+1] = cmmdc(gcdsecv[i][j], V_{j+1})$. O observație importantă este că $gcdsecv[i][j+1]$ este, fie $gcdsecv[i][j]$ (dacă V_{j+1} se divide cu $gcdsecv[i][j]$), fie **cel mult jumate** din $gcdsecv[i][j]$ (2).

O altă observație importantă este următoarea: știm că $gcdsecv[i][j] = cmmdc(gcdsecv[i][j-1], V_j)$. Dacă $gcdsecv[i-1][j-1] = gcdsecv[i][j-1]$, atunci și $gcdsecv[i-1][j] = gcdsecv[i][j]$ (3).

Având în vedere cele două observații, vom itera prin șir de la dreapta spre stânga, și vom păstra doar a doua dimensiune a lui $gcdsecv$. Presupunem că avem fixată poziția i . Inițial, $gcdsecv[i] = V_i$. Vom itera de la i spre N , iar, dacă $gcdsecv[j]$ **nu** își modifică valoarea, atunci, conform observației (3), nici valorile $gcdsecv[j']$, $j' > j$ nu se vor modifica. În caz contrar, conform observației (2), fiecare valoare din $gcdsecv$ se poate actualiza de maxim $\log(V_{max})$ ori.

Pentru a afla numărul de subsecvențe canonice care au capătul stâng fixat în i , vom căuta binar peste $gcdsecv$, unde este ultima poziție astfel încât $gcdsecv[j] = V_i$. Complexitate finală $O(N(\log^2(V_{max}) + \log(N)))$.

Soluția alternativă pentru $C = 1$ - Andrei Lețu

O soluție liniară pentru cerința 1 se bazează pe următoarea observație: considerăm subsecvența canonică de lungime maximă, cu capătul stâng fixat în poziția i . Fie această subsecvență $V_i, V_{i+1}, \dots, V_j$. Fie k o poziție din intervalul $i \dots j$. Cum V_k se divide cu V_i , $V_k \geq V_i$, iar subsecvența $V_i, \dots, V_j$ este subsecvență canonică de lungime maximă, cu capătul stâng fixat în i , atunci, subsecvența canonică cu capătul stâng fixat în k va avea capătul drept în poziția $j' \leq j$, deci va fi inclusă complet în intervalul $i \dots j$ (4).

Folosind observația de mai sus, vom itera prin șir de la stânga la dreapta, și vom ține mereu o poziție st . Inițial, $st = 1$. Dacă V_i se divide cu V_{st} , atunci, nu vom modifica indicele. În caz contrar, deducem că nu mai putem extinde subsecvența canonică care are capătul stâng fixat în st , deci vom actualiza $st = i$. Aceasta soluție are complexitate $O(N)$.

Soluție alternativă - Mircea Măierean

Ne vom folosi de observația de la soluția anterioară. Pentru fiecare poziție i , vom reține cea mai din dreapta poziție j astfel încât $1 \leq i \leq j \leq n$, iar întreaga subsecvență $i, i+1, i+2, i+3, \dots, j$ este canonică. Reținem aceste poziții într-un vector $next$, unde $next[i] = j$.

Pentru a avea în acest vector pozițiile corespunzătoare, vom itera de la dreapta la stânga. Pentru fiecare poziție i , inițializăm $next[i] = i$. După aceea, ne uităm la valoarea în vector aflată la poziția $next[i] + 1$. Dacă $a[next[i] + 1]$ este divizibil la $a[i]$, înseamnă că cele două subsecvențe, $[i \dots next[i]]$ și $[next[i] + 1 \dots next[next[i]]]$, reunite, formează de asemenea o subsecvență canonică. Așadar, cât timp se respectă această condiție, actualizăm valoarea lui $next[i]$, aceasta devenind $next[next[i] + 1]$. Dintre acestea, pentru prima cerință vom selecta lungimea maximă, iar pentru a doua, vom aduna lungimile tuturor subsecvențelor maxime. Soluția are complexitate $O(N)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Student **Alex-Matei Ignat**, Universitatea din București, Facultatea de Matematică și Informatică, București
- Programator **Alexandra Nicola**, Google, București
- Asistent doctor **Alexandru Ioniță**, Universitatea "Alexandru Ioan Cuza", Facultatea de Informatică, Iași
- Student **Alexandru-Mihai Lușcan**, Universitatea "Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca
- Profesor **Alice Georgescu**, Colegiul Național "Mihai Viteazul", Ploiești
- Student **Andrei-Cristian Ivan**, Universitatea Națională de Științe și Tehnologie Politehnica, Facultatea de Automatică și Calculatoare, București

- Student **Andrei Lețu**, Universitatea "Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca
- Profesor **Cristina Anton**, Colegiul Național "Gheorghe Munteanu Murgoci", Brăila
- Student **David Vișan**, Southern University of Denmark, Sønderborg
- Programator **Mihai Gheorghe**, Academia de Studii Economice, Facultatea de Cibernetică, București
- Student **Mircea Măierean**, ETH, Zurich
- Lector doctor **Paul Diac**, Universitatea "Alexandru Ioan Cuza", Facultatea de Informatică, Iași
- Student **Raul Ardelean**, Universitatea "Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca
- Student **Robert Moldovan**, Universitatea "Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca
- Student **Sebastian Predescu**, Universitatea Națională de Științe și Tehnologie Politehnica, Facultatea de Automatică și Calculatoare, București
- Student **Victor Botnaru**, Universitatea Națională de Științe și Tehnologie Politehnica, Facultatea de Automatică și Calculatoare, București