Olimpiada Națională de Informatică
Etapa națională, Clasa a VII-a
Descrierea soluțiilor

Baia Mare, 23 Martie 2026

Comisia științifică

Problema 1. Mapat

Propusă de: stud. Mitri Robert-Cristian, Universitatea Națională de Științe și Tehnologie Politehnica, Facultatea de Automatică și Calculatoare, București

Pentru fiecare query cu parametrii L și i, construim vectorul A, unde $A[j]$ reprezintă valoarea mapatului cu colțul principal (i, j) și lungimea L, pentru fiecare j cu $L \leq j \leq M$. Conform definiției:

$$A[j] = v[i][j] \cdot \sum_{x=i-L+1}^{i} \sum_{y=j-L+1}^{j} v[x][y].$$

Răspunsul la întrebările de tipul 2 se obține în toate soluțiile prin calcularea unei funcții $f(x)$, definită ca numărul de subsecvențe $[st, dr]$ cu suma $\leq x$, iar răspunsul final este $f(maxVal) - f(minVal - 1)$. Deoarece $v[i][j] \geq 0$, nu există subsecvențe cu suma negativă, deci $f(x) = 0$ pentru $x < 0$; acest caz apare când $minVal = 0$.

Soluție Brute Force — 34 de puncte

Suma S a fiecărui mapat se calculează parcurgând explicit toate cele L^2 celule, deci construirea vectorului A costă $O(M \cdot L^2)$ per query. Funcția $f(x)$ se evaluează în $O(M^2)$ prin parcurgerea tuturor perechilor (st, dr) și calculul direct al sumei subsecvenței. Similar, răspunsul la întrebările de tipul 1 se obține în $O(M^2)$ iterând st și extinzând dr cât timp suma rămâne $\leq maxVal$.

Complexitatea totală este $O\big((M \cdot L^2 + M^2) \cdot (q + p)\big)$, obținând **34 de puncte**.

Soluție Optimizată — până la 75 de puncte

Se aduc două optimizări independente față de soluția Brute Force.

Prima optimizare vizează calculul lui $A[j]$. Precalculând matricea sumelor parțiale 2D, $P[i][j] = \sum_{x=1}^{i} \sum_{y=1}^{j} v[x][y]$, suma oricărui subdreptunghi se obține în $O(1)$:

$$S = P[i][j] - P[i-L][j] - P[i][j-L] + P[i-L][j-L].$$

Construirea vectorului A devine astfel $O(M)$ per query, după o precalculare de $O(N \cdot M)$.

A doua optimizare vizează numărarea subsecvențelor. Construim sumele parțiale $pref[j] = A[L] + \cdots + A[j]$ (cu $pref[L-1] = 0$). Deoarece $A[j] \geq 0$, vectorul $pref$ este crescător, deci pentru un st fix putem căuta binar cel mai mare dr cu $pref[dr] \leq pref[st-1] + x$. Iterând peste toate valorile lui st, evaluarea lui $f(x)$ costă $O(M \log M)$.

Complexitatea totală este $O\big(N \cdot M + M \log M \cdot (q+p)\big)$, obținând **până la 75 de puncte**.

Soluție cu Two Pointers — 100 de puncte

Factorul logaritmic din soluția anterioară se elimină observând că $pref$ este monoton crescător, permițând înlocuirea căutării binare cu tehnica celor doi pointeri.

Menținem doi indici st și dr, inițializați la L, și o variabilă sum reprezentând suma curentă $A[st] + \cdots + A[dr]$, actualizată în $O(1)$: la incrementarea lui dr adunăm $A[dr]$, iar la incrementarea lui st scădem $A[st]$. Incrementăm dr; cât timp sum depășește x, incrementăm st. La fiecare pas, numărul de subsecvențe valide care se termină în dr este $dr - st + 1$, care se adaugă la $f(x)$. Fiecare pointer parcurge cel mult M poziții, deci evaluarea lui $f(x)$ costă $O(M)$.

Pentru întrebările de tipul 1, lungimea maximă a unei subsecvențe cu suma $\leq maxVal$ este maximul valorilor $dr - st + 1$ obținute pe parcursul aceluiași proces.

Complexitatea totală este $O\big(N \cdot M + M \cdot (q+p)\big)$, obținând **100 de puncte**.

Soluție alternativă — 100 de puncte

Propusă de Jonathan Mogovan

Soluția anterioară recalculează vectorul A în $O(M)$ pentru fiecare întrebare. Dacă mai multe interogări vizează aceeași pereche de parametri (i, L), acest calcul devine inutil.

Inovația constă în gruparea interogărilor înainte de a le rezolva: le citim pe toate de la început, le reținem poziția inițială și le sortăm după parametrii (i, L). Astfel, interogările care depind de același vector A ajung una lângă alta, permițând calcularea acestuia o singură dată.

Pentru fiecare grup cu aceiași (i, L), construim vectorul A o singură dată, apoi răspundem la toate interogările din grup folosind tehnica *Two Pointers*. La final, afișăm rezultatele în ordinea inițială.

Fie U numărul de perechi distincte (i, L) din datele de intrare. Pe testele în care interogările se repetă pe aceleași zone, valoarea lui U este semnificativ mai mică decât numărul total de întrebări (suma $q + p$). Datorită acestui fapt, timpul total alocat generării vectorului A scade masiv, de la $O\big((q+p) \cdot M\big)$ la doar $O(U \cdot M)$.

Complexitatea totală devine:

$$O\big(N \cdot M + (q+p)\log(q+p) + U \cdot M + (q+p) \cdot M\big)$$

Această abordare elimină complet operațiile inutile și obține, de asemenea, 100 de puncte.

Problema 2. Tricouri

Propusă de: stud. Cotoi Rareș-Andrei, Universitatea „Babeș-Bolyai", Cluj-Napoca,
prof. Popa Daniel, Colegiul Național „Aurel Vlaicu", Orăștie

Cerința 1.

O posibilă soluție este ca la fiecare pas să parcurgem șirul de la stânga la dreapta și să căutăm prima poziție i cu proprietatea $a[i] < a[i+1]$. Eliminăm cifra $a[i]$, deoarece aceasta ne împiedică să plasăm o cifră mai mare pe o poziție mai semnificativă. Dacă nu există o astfel de poziție (șirul este descrescător), eliminăm ultima cifră. Întrucât trebuie să eliminăm K cifre, va trebui să repetăm acest proces de K ori, deci complexitatea este $O(N \cdot K)$. Această abordare obține 14 puncte pentru $C = 1$.

Pentru rezolvarea optimă a acestei cerințe, vom folosi o structură de tip stivă. Parcurgând cifrele de la stânga la dreapta, pentru fiecare cifră curentă $a[i]$:

- cât timp stiva nu este goală, mai avem eliminări disponibile ($K > 0$) și dacă cifra din vârful stivei este mai mică decât $a[i]$, scoatem elementul din vârf (efectuăm o eliminare, deci K scade cu 1);
- adăugăm cifra $a[i]$ în stivă.

Dacă, după parcurgerea tuturor cifrelor, mai rămân eliminări neefectuate ($K > 0$), eliminăm din vârful stivei K cifre. La final, cifrele rămase în stivă, citite de la bază la vârf, formează valoarea tactică (eliminăm eventualele zerouri nesemnificative de la început). Cum fiecare cifră este adăugată și eliminată din stivă cel mult o dată, complexitatea totală este $O(N)$. Această soluție obține 40 de puncte, pentru toate subtask-urile cu $C = 1$.

Cerința 2.

O primă observație este faptul că, dacă ținem minte valoarea tactică a echipei într-o variabilă de tip `int` și afișăm stabilitatea folosind funcția `sqrt`, vom obține 10 puncte pentru $C = 2$. De asemenea, dacă valoarea tactică este ținută într-o variabilă de tip `long long`, obținem încă 10 puncte adiționale. Totuși, pentru restul restricțiilor, este nevoie să ținem minte valoarea tactică ca un șir de cifre, sub forma unui tablou unidimensional.

Știind că V este valoarea tactică, vom construi răspunsul stabilitatea echipei $p = \lfloor \sqrt{V} \rfloor$ cifră cu cifră, de la cea mai semnificativă la cea mai puțin semnificativă. Dacă V are D cifre, atunci p are exact $\lceil D/2 \rceil$ cifre (deoarece $10^{D-1} \leq V < 10^D$ implică $10^{(D-1)/2} \leq \sqrt{V} < 10^{D/2}$). Observăm că fiecare cifră a rezultatului p influențează exact două cifre ale lui V (de exemplu, dacă $p = 98$, atunci $p^2 = 9604$: cifra zecilor (9) influențează primele două cifre ale pătratului, iar cifra unităților (8) le influențează pe ultimele două). Vom grupa cifrele lui V în perechi de la dreapta la stânga (completând cu un zero la stânga dacă D este impar) și menținem două valori (numere mari):

- p - rezultatul parțial (primele k cifre ale lui $[\sqrt{V}]$, după pasul k);
- c - restul curent.

După procesarea primelor k perechi de cifre, notăm cu V_k numărul format din primele $2k$ cifre ale lui V. La fiecare pas, trebuie să avem $p^2 \leq V_k$ și $c = V_k - p^2$ (adică p este cel mai mare număr de k cifre al cărui pătrat nu depășește V_k, iar c este restul rămas după scădere). La pasul $k+1$, aducem următoarea pereche de cifre d (cu $0 \leq d \leq 99$). Noul număr format din primele $2(k+1)$ cifre este $V_{k+1} = V_k \cdot 100 + d$, iar noul rezultat parțial va fi $p' = p \cdot 10 + x$, unde $x \in \{0, \ldots, 9\}$ este cifra pe care o căutăm. Trebuie să alegem cel mai mare x astfel încât $(p')^2 \leq V_{k+1}$. Calculăm noul rest: $c' = V_{k+1} - (p')^2 = (100c + d) - (20p + x) \cdot x$. Condiția $c' \geq 0$ (echivalentă cu $(p')^2 \leq V_{k+1}$) devine: $(20p + x) \cdot x \leq 100c + d$. Observăm că $100c + d$ este exact valoarea lui c după ce am adus noua pereche (înmulțim restul vechi cu 100 și adunăm perechea). Așadar, la fiecare pas:

1. Actualizăm restul: $c = c \cdot 100 + d$;
2. Căutăm cel mai mare x cu $(20p + x) \cdot x \leq c$;
3. Actualizăm: $c = c - (20p + x) \cdot x$ și $p = p \cdot 10 + x$.

Păstrăm noul rest, care este $c' = V_{k+1} - (p')^2 \geq 0$, iar p' este cel mai mare număr de $k+1$ cifre cu $(p')^2 \leq V_{k+1}$. Expresia $(20p + x) \cdot x$ se poate rescrie ca $20p \cdot x + x^2$. Operațiile necesare sunt:

- $20p$: se calculează o singură dată per pas (înmulțim p cu 10, apoi cu 2);
- $20p \cdot x$: înmulțirea unui număr mare cu cifra x;
- Adunarea lui x^2 (≤ 81): adunarea unui număr mic la un număr mare;
- Comparația cu c.

Apoi, verificăm cifrele $x = 0, 1, \ldots, 9$ în ordine; ne oprim la primul x pentru care condiția nu mai este satisfăcută (deoarece $(20p + x) \cdot x$ este strict crescătoare în x). Cifra corectă este ultima care a satisfăcut condiția.
Algoritmul procesează $D/2$ perechi. La pasul k, numerele p și c au k cifre, iar operațiile (înmulțire cu o cifră, comparare, scădere) au complexitatea $O(k)$. Așadar, complexitatea algoritmului este $O\left(\frac{D^2}{4}\right) = O(D^2)$, soluție care obține 87 de puncte (împreună cu cerința 1) și folosește operațiile de adunare, scădere și înmulțire cu scalar a numerelor mari.
Pentru ultimele 13 puncte, toate cifrele sunt doar 0 sau 1, iar valoarea tactică se interpretează ca un număr în baza 2 (cu până la 18000 de cifre). Convertirea numărului din baza 2 în baza 10 și apoi extragerea radicalului în baza 10 este prea lentă și nu obține punctajul maxim.
Observația cheie este faptul că algoritmul descris mai sus funcționează în orice bază, nu doar în baza 10. Aplicându-l în baza 2, singurele diferențe față de varianta în baza 10 sunt:

- grupăm cifrele în perechi de cifre în baza 2 (nu în baza 10);
- restul se actualizează cu $c = 4c + d$ (în loc de $c = 100c + d$);
- cifra următoare a rezultatului este fie 0, fie 1 (nu trebuie să căutăm printre $0, \ldots, 9$): verificăm doar dacă $4p + 1 \leq c$: dacă da, $c = c - (4p + 1)$ și $p = 2p + 1$; altfel, $p = 2p$;
- reținem p și c ca numere mari în baza 10, deci rezultatul p este direct în zecimal.

Complexitatea este $O\left(\frac{B^2}{4}\right)$ (unde B este numărul de cifre în baza 2), mai rapidă decât conversia la baza 10 urmată de extragerea radicalului în baza 10. Această soluție obține 100 de puncte.

Soluție alternativă - Trecerea la baza 10^9 (100 de puncte)

Propusă de Jonathan Mogovan și Bogdan Borodi

O abordare mult mai eficientă și elegantă se obține schimbând complet modul în care reprezentăm numerele mari. Vom grupa câte 9 cifre zecimale într-o singură variabilă normală pe 32 de biți. Practic, transformăm numărul mare într-unul scris în baza 10^9.

Această comprimare reduce lungimea șirurilor de 9 ori, iar algoritmul de extragere a radicalului devine mult mai rapid, deoarece la fiecare pas vom coborî câte două blocuri a câte 9 cifre, adică 18 cifre deodată. În mod firesc, și rezultatul se va construi tot bloc cu bloc. În loc să căutăm liniar o simplă cifră de la 0 la 9, acum vom determina o cifră a rezultatului între 0 și $10^9 - 1$. Cum o căutare secvențială ar ieși din timpul de execuție, vom folosi o căutare binară pentru a identifica rapid valoarea corectă.

Prin această grupare, lungimea numerelor mari scade dramatic (notăm noua lungime cu $L \approx D/9$, unde D este numărul inițial de cifre). Deși căutarea binară adaugă un factor de $\log_2(10^9)$ la fiecare pas, numărul total de operații scade considerabil: de la o proporție de $O(D^2)$ în soluția de bază, ajungem la aproximativ $O(L^2 \cdot 30)$, ceea ce înseamnă aproximativ $\frac{D^2}{81} \cdot 30 \approx \frac{D^2}{2.7}$. În practică, reducerea acestui factor scade semnificativ timpul de execuție.

De asemenea, avantajul acestei optimizări este eliminarea completă a tratării separate pentru subtask-ul 6. Deoarece algoritmul nostru de bază a devenit suficient de rapid, nu mai avem nevoie de un algoritm dedicat extragerii radicalului pentru numere binare. Dacă șirul inițial este format exclusiv din 0 și 1, pur și simplu îl convertim direct în baza 10^9 și aplicăm exact același algoritm cu căutare binară. Această soluție obține, de asemenea, 100 de puncte.

Problema 3. Alchimie

Propusă de: stud. Rotaru Răzvan-Alexandru, Universitatea „Alexandru Ioan Cuza", Facultatea de Informatică, Iași

Pentru a rezolva problema, trebuie să abordăm distinct cele două cerințe pe baza parametrului C. Cerința 1 vizează determinarea unei secvențe de lungime maximă formată din „Perechi Pure". Două eprubete vecine formează o astfel de pereche dacă cantitățile lor sunt prime între ele, adică $\gcd(V_i, V_{i+1}) = 1$. Cerința 2 necesită obținerea „Scării Perfecte", o configurație în care eprubeta i are valoarea i, folosind operații de rezonanță *adun* și *scad* asupra elementelor adiacente. Toate valorile trebuie să rămână strict în intervalul $[1, 100\,000]$, încadrate într-o limită de 1 000 000 de operații. Am notat cu V_{max} cea mai mare valoare din sirul V.

Subtask 1 ($C = 1$, $N, V_{max} \leq 200$) — 13 puncte

Pentru acest subtask cu limite reduse, putem construi o soluție *brute force*. Calculăm pentru fiecare pereche de numere consecutive (A, B) cel mai mare divizor comun printr-o parcurgere liniară de la 1 la $\min(A, B)$. După acest calcul, iterăm prin toate secvențele posibile și verificăm dacă sunt alcătuite exclusiv din perechi pure. O reținem pe cea mai lungă și o afișăm la final.

Complexitatea totală a acestei abordări este $O(N^3 + N \cdot V_{max})$.

Subtask 2 ($C = 1$, $N \leq 10\,000$, $V_{max} \leq 100\,000$) — 16 puncte

Calculul naiv devine ineficient pentru limitele mai mari. Optimizarea se face pe două fronturi:

1. Pentru evaluarea rapidă a *cmmdc(A, B)*, vom folosi Algoritmul lui Euclid, reducând timpul la $O(\log V_{max})$.
2. Pentru găsirea secvenței de lungime maximă, eliminăm iterarea $O(N^3)$ aplicând tehnica *Two Pointers*. Menținem un interval $[st, dr]$ care se extinde la dreapta atâta timp cât $\text{cmmdc}(V_{dr-1}, V_{dr}) = 1$. Când condiția nu mai este respectată, mutăm capătul *st* la poziția curentă și reluăm procesul.

Complexitatea totală scade drastic la $O(N \log V_{max})$.

Subtask 3 ($C = 2$, $V_{max} = 1$) — 11 puncte

Având în vedere că șirul inițial este format exclusiv din valoarea 1, putem forma „Scara Perfectă" de la stânga la dreapta într-un număr minim de pași. Se aplică operația *adun i i − 1* pentru fiecare i de la 2 la N.

Exemplu de transformare: $(1, 1, 1, 1, 1) \to (1, 2, 1, 1, 1) \to (1, 2, 3, 1, 1) \to \cdots \to (1, 2, 3, 4, 5)$.

Timpul de execuție este $O(N)$, iar numărul de operații este exact $N - 1$.

Subtask 4 ($C = 2$, $N, V_{max} \leq 200$, **există măcar un** $V_i = 1$) — 11 puncte

Prezența unei valori de 1 în șir acționează ca un instrument de „distrugere" pentru elementele mari. Deoarece $V_{max} \leq 200$, putem folosi acea valoare de 1 pentru a scădea unități din vecinii ei (prin operația *scad*) până când aceștia ajung la rândul lor la valoarea 1. Propagăm acest efect în stânga și în dreapta, extinzând secvența de 1-uri până când întregul șir devine plin de 1. La final, aplicăm direct soluția liniară de la Subtaskul 3.

Subtask 5 ($C = 2$, $N, V_{max} \leq 200$) — 14 puncte

Problema garantează existența a cel puțin unei Perechi Pure în șirul inițial. Fie (A, B) această pereche, având $cmmdc(A, B) = 1$. Putem transforma această secvență în $(1, 1)$ printr-o serie de scăderi repetate ale elementului mai mic din cel mai mare. Acest proces imită exact pașii Algoritmului lui Euclid prin scăderi.

Exemplu: $(8, 19) \to (8, 11) \to (8, 3) \to (5, 3) \to (2, 3) \to (2, 1) \to (1, 1)$.

După obținerea bazei $(1, 1)$, aplicăm raționamentul de la Subtaskul 4, extinzând secvența de 1-uri spre stânga și spre dreapta, pentru a finaliza rezolvarea.

Subtask 6 ($C = 2$, $N \leq 10\,000$, $V_{max} \leq 1\,000$) — 13 puncte

Pentru valori de până la 1 000, scăderea naivă repetată cu 1 ar consuma prea multe operații. Scopul este minimizarea numărului de pași necesari pentru a reduce secvența $(1, 1, X)$ la $(1, 1, 1)$.

Soluția este să incrementăm prima eprubetă de 1 de lângă X până atinge o valoare $B \approx [\sqrt{X}]$. Acum, în loc să scădem 1, scădem din X valoarea B de mai multe ori. După ce restul lui X devine C, un numar mai mic decât B, reducem B cu o unitate (folosind celălalt 1 din stânga lui) pâna când ajungem la $C - 1$ (numai în cazul in care C este diferit de 1). În acest mod obținem secventa $(1, C-1, C)$. Printr-o operație de scadere aducem secvența la $(1, C-1, 1)$. În continuare vom scadea 1 din elementul $C - 1$ pana cand va ajunge la 1 Această tehnică blochează numărul de operații per element la un numar mai mic sau egal cu $3\sqrt{X}$, asigurând un total mai mic de $V_{max} + (N - 2) * 3 * \sqrt{X}$ operații.

Subtask 7 ($C = 2$, $N \leq 10\,000$, $V_{max} \leq 100\,000$) — 22 de puncte

Pentru limitele maxime, optimizarea cu radical este insuficientă. Inovația necesară pentru a garanta cele 1 000 000 de operații constă în exploatarea **Șirului lui Fibonacci**.

Găsim prima pereche pură și o transformăm în $(1, 1)$ prin scăderi repetate, la fel ca la Subtaskul 5. Această pereche devine punctul de plecare. De aici, vom extinde această secvență de valori de 1 pas cu pas, aducând pe rând fiecare element adiacent secvenței la valoarea 1.

Pentru a reduce eficient un element curent țintă, ne folosim de cele două eprubete cu valoarea 1 aflate imediat lângă el. Creștem exponențial valorile acestor două eprubete ajutătoare: adunând constant eprubeta mai mică la cea mai mare, generăm secvența Fibonacci: $(1, 2) \rightarrow (3, 2) \rightarrow (3, 5) \rightarrow \ldots$, oprindu-ne doar când suma lor ar depăși valoarea țintei. Cel mai mare număr Fibonacci $\leq 100\,000$ se obține în doar $\sim 25(\log_\phi)$ operații!

Apoi, scădem din țintă eprubeta alăturata (care acum conține un număr Fibonacci mare) de câte ori este posibil. Când valoarea țintei devine mai mică decât acest număr Fibonacci, trebuie să „coborâm" pe scara Fibonacci.

Partea elegantă a acestei metode este că **nu avem nevoie de o structură de date suplimentară (precum o stivă)** pentru a reține operațiile. Datorită proprietății șirului lui Fibonacci ($F_k = F_{k-1} + F_{k-2}$), pentru a face pasul înapoi, este suficient să scădem eprubeta mai mică din eprubeta mai mare. Astfel, prin aplicarea unei operații de *scad* între cele două eprubete ajutătoare, valorile (F_k, F_{k-1}) redevin (F_{k-2}, F_{k-1}).

Repetăm alternativ acest proces (scădem numărul Fibonacci curent din țintă atâta timp cât putem, apoi coborâm un pas pe scara Fibonacci prin scăderea eprubetelor între ele) până când secvența redevine $(1, 1)$. Eventualul rest din eprubeta țintă este redus rapid prin scăderi cu unități până ajunge exact la 1. Astfel, elementul țintă a devenit 1, iar secvența noastră s-a lungit cu încă o poziție.

Procesul se reia identic, extinzând secvența de 1-uri element cu element în stânga și în dreapta, până când absolut tot șirul este format doar din valoarea 1. La final, șirul $(1, 1, \ldots, 1)$ este transformat în „Scara Perfectă" printr-o parcurgere simplă $O(N)$, așa cum este descris la Subtaskul 3.

Complexitatea totală a operațiilor este redusă la aproximativ $V_{max} + (N - 2) * 3 * \log_\phi V_{max}$, menținând numărul de rezonanțe cu mult sub limita impusă de problemă.

Soluție alternativă - 100 de puncte

Propusă de: prof. Adrian Panaete

O abordare diferită se bazează pe reducerea treptată a valorilor maxime din șir folosind puteri ale lui 2.

În loc să izolăm o pereche de valori $(1, 1)$ încă de la început, vom impune o limită superioară T pentru toate elementele șirului. Pornim cu $T = 2^{16} = 65\,536$ (deoarece acoperă valoarea maximă posibilă) și, la fiecare pas, obiectivul este să modificăm șirul astfel încât toate valorile să devină strict mai mici decât T. Având deja garantat de la pasul anterior că valorile sunt $< 2T$, vom înjumătăți repetat pragul ($T \leftarrow T/2$) până când ajungem la $T = 2$.

Reducerea valorilor pentru un prag T fixat se realizează în trei faze:

1. Izolarea unei valori mici: Căutăm sau construim (printr-un set de analize de caz aplicate pe primele trei poziții) o singură valoare strict mai mică decât T la un anumit indice q.

2. Obținerea unei perechi: Folosind elementul de pe poziția q, modificăm unul dintre vecinii săi (stânga sau dreapta) printr-o succesiune scurtă de adunări și scăderi, astfel încât și acesta să devină strict mai mic decât T.

3. Propagarea limitei: Având acum două valori vecine $< T$, le putem folosi ca bază pentru a reduce restul elementelor. Extindem intervalul de valori bune parcurgând șirul o dată către stânga și o dată către dreapta. Când întâlnim un element țintă $V_p \geq T$, ne folosim de cei doi vecini adiacenți (deja $< T$). Pentru a reduce eficient elementul țintă, le creștem valorile adunându-le repetat (cea mai mică la cea mai mare, generând o creștere proporțională cu șirul lui Fibonacci). Ne oprim când vecinii devin suficient de mari pentru a coborî valoarea țintei sub pragul T din una sau două scăderi.

Procesul se repetă până când pragul coboară la $T = 2$. Din regulile de siguranță ale problemei știm că nicio eprubetă nu poate conține mai puțin de 1 Esență. Cum toate valorile trebuie să fie strict mai mici decât 2, rezultă că întregul șir va fi format exclusiv din valori de 1.

Din configurația $(1, 1, \ldots, 1)$, „Scara Perfectă” se obține trivial, parcurgând șirul de la stânga la dreapta și aplicând operația `adun i i-1` pentru fiecare i de la 2 la N.

Un detaliu tehnic elegant, care poate fi aplicat oricărei soluții (indiferent de algoritmul ales), constă în optimizarea memoriei pentru reținerea operațiilor. Deoarece numărul maxim de pași poate ajunge la 1 000 000, stocarea sub formă de șiruri de caractere sau structuri complexe consumă inutil memorie și timp. În schimb, fiecare comandă poate fi comprimată direct într-un singur număr întreg folosind operații pe biți: biții superiori memorează poziția țintă i, penultimul bit indică poziția vecinului ajutător ($i - 1$ sau $i + 1$), iar ultimul bit indică tipul operației (adunare sau scădere). La afișare, informația este extrasă rapid prin decodificarea acestor biți.

Echipa

Problemele pentru această etapă au fost pregătite de:

- prof. Costineanu Veronica-Raluca, Colegiul Național „Ștefan cel Mare", Suceava

- prof. Panaete Adrian, Colegiul Național „A.T. Laurian", Botoșani
- prof. Popa Daniel, Colegiul Național „Aurel Vlaicu", Orăștie
- prof. Rotar Dorin-Mircea, Colegiul Național „Samuil Vulcan", Beiuș
- stud. Cotoi Rareș-Andrei, Universitatea „Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca
- stud. Mogovan Jonathan, Universitatea „Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca
- stud. Rotaru Răzvan-Alexandru, Universitatea „Alexandru Ioan Cuza", Facultatea de Informatică, Iași
- stud. Mitri Robert-Cristian, Universitatea Națională de Științe și Tehnologie Politehnica, Facultatea de Automatică și Calculatoare, București
- stud. Borodi Bogdan, Universitatea „Babeș-Bolyai", Facultatea de Matematică și Informatică, Cluj-Napoca