Olimpiada Națională de Informatică, Clasa a VIII-a Descrierea soluțiilor

Comisia științifică

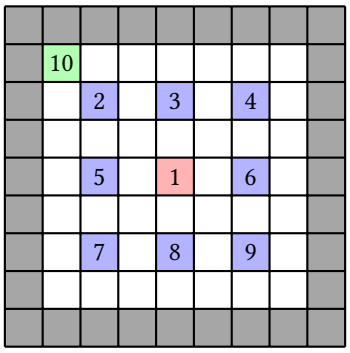
March 23, 2026

Problema 1. Bafta

Propusă de: stud. Alin Gabriel Răileanu, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași

Structura

Pentru început, să analizăm cum se așază elevii în sală. Fiecare elev alege celula care maximizează distanța față de cea mai apropiată celulă ocupată (supraveghetor sau alt elev).
Să analizăm ordinea de umplere pentru $N = 3$ (matrice 9×9, cu $7 \times 7 = 49$ celule interioare):



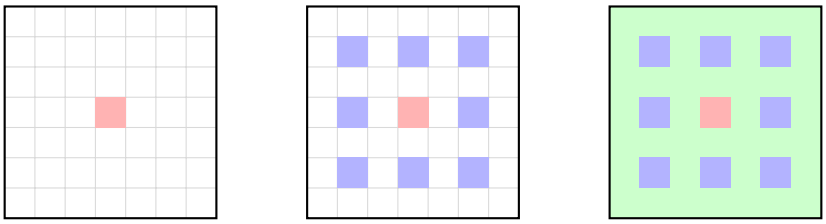
Observăm că elevii se așează în **niveluri** concentrice:

- **Nivelul 1**: centrul matricii (1 celulă) – distanța maximă față de margine
- **Nivelul 2**: $3 \times 3 - 1 = 8$ celule noi – la jumătatea distanței
- **Nivelul 3**: $7 \times 7 - 9 = 40$ celule noi – ș.a.m.d.

Generalizare: La nivelul l, avem o grilă de dimensiune $(2^l - 1) \times (2^l - 1)$. Numărul de celule noi la acest nivel este:

$$(2^l - 1)^2 - (2^{l-1} - 1)^2$$

Fie $\text{side}[l] = 2^l - 1$. Primele $\text{side}[l]^2$ celule ocupate formează o grilă completă până la nivelul l.



Nivel 1
$1^2 = 1$ celulă

Nivel 2
$3^2 = 9$ celule

Nivel 3
$7^2 = 49$ celule

Subtask 1 (16 puncte) – Simulare directă

Restricții: $2 \leq N \leq 4$, $Q = 5$
Soluție: Simulăm procesul de așezare. Pentru fiecare elev k de la 1 la K:

1. Calculăm pentru fiecare celulă liberă distanța minimă față de celulele ocupate
2. Alegem celula cu distanța minimă maximă (la egalitate, cea mai mică linie, apoi coloană)
3. Marcăm celula ca ocupată

Complexitate timp: $O(Q \cdot K \cdot M^2)$ unde $M = 2^N$.
Complexitate memorie: $O(M^2)$.

Subtask 2 (24 puncte) – Simulare cu matrice

Restricții: $4 < N \leq 10$, $Q = 5$
Observație: Nu trebuie să calculăm distanța pentru fiecare celulă. Din structură, știm că:

- Primul elev se așează în centrul matricei: $(2^{N-1}, 2^{N-1})$
- La un nivel cu distanță d între celulele ocupate pe acest nivel, parcurgem toate pozițiile (i, j) cu $i, j \in \{d, 2d, 3d, \ldots\}$ care nu sunt încă ocupate

Algoritm:

1. Inițializăm matricea cu 0 (neocupat)
2. Plasăm primul elev în centru: $a[2^{N-1}][2^{N-1}] = 1$
3. Pentru fiecare distanță d:

- Parcurgem toate pozițiile (i, j) cu $i, j \in \{d, 2d, \ldots\}$
- Dacă $a[i][j] = 0$, plasăm următorul elev și incrementăm contorul
- Ne oprim când contorul ajunge la K

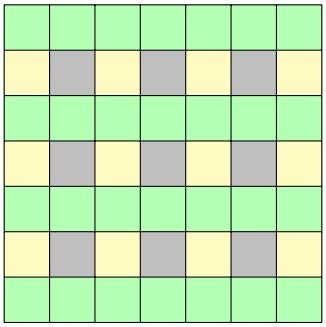
Complexitate timp: $O(K)$ per query.
Complexitate memorie: $O(M^2)$, unde $M = 2^N$.
Optimizare nenecesară: Ordinea de umplere este **aceeași** indiferent de N. Mai exact, dacă elevul k se așează la poziția (r, c) într-o matrice cu $N = n_1$, atunci într-o matrice cu $N = n_2 > n_1$, elevul k se așează la poziția $(r \cdot 2^{n_2-n_1}, c \cdot 2^{n_2-n_1})$.
Astfel:

1. Precalculăm pozițiile pentru $N = 10$ o singură dată prin procedeul de mai sus
2. Pentru fiecare query (N, K): scalăm coordonatele cu factorul 2^{N-10}.

Subtask 3 (20 puncte) – Generare "virtuală"

Restricții: $K \leq 10^6$, $Q = 5$
Pentru $N > 10$, nu putem stoca matricea, dar $K \leq 10^6$, deci putem genera pas cu pas ocuparea acesteia.
Algoritm:

1. Pentru fiecare nivel cu distanță d, parcurgem rândurile cu pas d
2. Pe rânduri impare (în cadrul nivelului), parcurgem coloanele cu pas d
3. Pe rânduri pare, parcurgem coloanele cu pas $2d$
4. Numărăm celulele până ajungem la K
5. Apoi generăm pentru nivelul următor cu distanță $\frac{d}{2}$



Nivelul 3: rânduri impare (verde, pas d) și pare (galben, pas $2d$)

Complexitate timp: $O(K)$ per query.
Complexitate memorie: $O(1)$.

Subtask 4 și 5 (40 puncte) – Formulă

Restricții: $Q \leq 10^3$ (Subtask 4), $Q \leq 10^5$ (Subtask 5)
Nu mai putem parcurge K celule pentru fiecare query când K poate fi 10^9.
Precalculare: Precalculăm side$[l]$ pentru $l \in \{0, 1, \ldots, 30\}$.
Pasul 1 – Determinarea nivelului: Găsim primul nivel l astfel încât side$[l]^2 \geq K$. Aceasta înseamnă că elevul K se așează la nivelul l.
Pasul 2 – Poziția în cadrul nivelului: Scădem celulele din nivelurile anterioare:

$$K \leftarrow K - \text{side}[l-1]^2$$

Acum K reprezintă numărul celulei în cadrul nivelului l.
Pasul 3 – Structura nivelului:
Să vizualizăm cum sunt aranjate celulele noi în nivelul 3 (side$[3] = 7$):

r1	1	2	3	4	5	6	7	pereche 0
r2	8		9		10		11	
r3	12	13	14	15	16	17	18	pereche 1
r4	19		20		21		22	
r5	23	24	25	26	27	28	29	pereche 2
r6	30		31		32		33	
r7	34	35	36	37	38	39	40	

Nivelul 3: celulele noi numerotate 1–40 (gri = niveluri anterioare)

În nivelul $l = 3$:

- Fiecare rând impar (r1, r3, r5, r7) conține cellsOddRow = side$[l] = 7$ celule
- Fiecare rând par (r2, r4, r6) conține cellsEvenRow = side$[l-1] + 1 = 4$ celule (doar coloanele impare)
- O pereche de rânduri (impar + par) conține cellsInOnePair = 7 + 4 = 11 celule

Pasul 4 – Determinarea coordonatelor locale:
Grupăm rândurile în perechi și analizăm structura unei perechi:

impar	1	2	3	4	5	6	7
par	8		9		10		11
	c1	c2	c3	c4	c5	c6	c7

O pereche de rânduri: celulele 1–11

1. Calculăm indexul perechii: pairIndex = $\lfloor K/\text{cellsInOnePair} \rfloor$
2. Calculăm poziția în pereche: $K \leftarrow K \mod \text{cellsInOnePair}$

3. Determinăm rândul și coloana:
    - Dacă $K = 0$: suntem pe ultima celulă a perechii anterioare (celula 11), deci rândul este $2 \cdot \text{pairIndex}$ și coloana este cellsOddRow (coloana 7)
    - Dacă $K \leq \text{cellsOddRow}$: suntem pe rândul impar (celulele 1–7), deci rândul este $2 \cdot \text{pairIndex} + 1$ și coloana este K
    - Altfel: suntem pe rândul par (celulele 8–11), deci rândul este $2 \cdot \text{pairIndex} + 2$ și coloana este $2 \cdot (K - \text{cellsOddRow}) - 1$ (coloanele impare: 1, 3, 5, 7)

Pasul 5 – Scalarea la coordonate reale: Factorul de scalare este $\text{scale} = \text{side}[N-l]+1 = 2^{N-l}$. Coordonatele finale sunt:

$$(\text{row} \cdot \text{scale}, \text{col} \cdot \text{scale})$$

Complexitate timp: $O(\log N)$ per query (dată de determinarea nivelului, restul pașilor fiind rezolvați în $O(1)$). Pentru subtask-ul 4 este suficientă o rezolvare în $O(\sqrt{K})$.
Complexitate memorie: $O(1)$.

În funcție de corectitudinea implementării și complexitate se vor obține punctaje parțiale.
Observație:
Problema poate fi rezolvată și în $O(1)$ timp per query, folosind operații pe biți pentru a determina nivelul direct, dar această optimizare nu este necesară pentru obținerea punctajului maxim.

Problema 2. Dj

Propusă de: stud. Eduard-Lucian Pîrțac, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași

Soluție subtask 2

Fie x primul element din șir.

Restricțiile acestui subtask garantează că șirul se poate egaliza folosind cel mult o operație per element. De aici rezultă că avem doar trei valori la care se poate egaliza șirul:

- x (nu aplicăm nicio operație);
- $x \cdot 2 + 1$ (aplicăm o operație de tip 1);
- câtul împărțirii lui x la 2 (aplicăm o operație de tip 2).

Încercăm pe rând câte o valoare dintre cele trei și verificăm dacă fiecare element din șir se poate aduce la această valoare prin cel mult o operație. În caz afirmativ, calculăm și numărul de operații asociate valorii respective, iar la final afișăm numărul minim de operații.

Soluție completă 1

Soluția are două părți: generarea numerelor posibile pentru un element din șir și găsirea răspunsului final. Fiecare parte are mai multe rezolvări.

Generarea numerelor

Mai întâi să facem câteva observații esențiale pentru un singur element, după care putem scala ideile la tot șirul:

1. Pornim de la o valoare x. Dacă aplicăm o operație de tip 1, valoarea devine $2 \cdot x + 1$. După, dacă aplicăm o operație de tip 2, valoarea devine $\lfloor \frac{2 \cdot x + 1}{2} \rfloor = x$ (câtul împărțirii lui $2 \cdot x + 1$ la 2 este x). Observăm astfel că secvența de operații $1, 2$ este mereu redundantă deoarece am plecat de la x și am ajuns tot la x, iar numărul de operații a crescut inutil. Prin urmare, pentru un element o strategie poate fi să aplicăm câteva operații de tip 2, iar apoi câteva operații de tip 1.

2. Dacă presupunem că egalizăm șirul la numărul y, cu y mai mare strict decât valoarea maximă din șir, singura variantă prin care elementele șirului puteau să ajungă la y este dacă erau $\frac{y-1}{2}$ la un moment în timp. Deci am fi putut să egalizăm șirul la $\frac{y-1}{2}$ cu număr mai mic de operații. Astfel, am demonstrat că numărul la care vom egaliza șirul nu va depăși niciodată valoarea maximă din șirul inițial.

3. Dacă pornim de la valoarea maximă $2 \cdot 10^5$ și aplicăm repetat operații de tip 2, vom ajunge la valoarea 0 în doar 18 operații. Similar, dacă pornim de la 0 și aplicăm repetat operații de tip 1, vom ajunge în 17 operații la cea mai mare valoare $\leq 2 \cdot 10^5$. Aceste calcule se pot face simplu pe hârtie sau pot fi demonstrate formal dacă se face asocierea dintre operațiile din enunț și reprezentarea binară a numerelor, 18 reprezentând numărul maxim de biți a unei valori.

Fiindcă numărul de operații de la ideea 3 este mic, putem să generăm toate valorile la care se poate ajunge dintr-un element din șir. Să notăm cu L numărul de astfel de valori.

Varianta 1

Pentru fiecare element, folosim o structură de tip coadă. Această coadă menține într-o poziție două valori (folosind un `struct`): numărul la care am ajuns și numărul de operații ca să ajungem la acest număr.

În coadă inserăm întâi toate valorile care pot fi obținute folosind zero sau mai multe operații de tip 2. Apoi, cât timp coada nu este goală, extragem câte un element și îi aplicăm o singură operație de tip 1. Dacă noua valoare nu apare deloc în coadă sau apare dar cu număr de operații mai mare, vom adăuga valoarea nouă în coadă.

Complexitatea timp pentru această variantă este în teorie $O(N \cdot L^2)$, dar în practică este mult mai rapidă, chiar mai rapidă decât varianta 2 de mai jos. Complexitatea spațiu este $O(N + L)$, deoarece suntem nevoiți să refolosim coada pentru fiecare element din șir pentru că nu avem memorie destulă pentru $N \cdot L$ elemente.

La final, coada o vom denumi ca *vectorul asociat* cu elementul din șir. Acești vectori îi vom folosi la găsirea răspunsului final.

Varianta 2

Pentru fiecare element, folosim două bucle `for` ca să parcurgem toate valorile de $0 \leq a \leq 18$ și $0 \leq b \leq 18$, unde a înseamnă câte operații de tip 2 aplicăm, iar b câte operații de tip 1. La fiecare pas calculăm noua valoare și adăugăm toate valorile într-un vector de dimensiune $L \leq 19 \cdot 19 = 361$ asociat cu câte un element din șir. Pentru fiecare număr din acest vector, trebuie să memorăm și numărul de operații $a + b$ pentru a ajunge la număr. Acest lucru îl putem face spre exemplu folosind un vector de tip `struct`.

Complexitatea timp pentru această variantă este $O(N \cdot L)$, iar complexitatea spațiu este $O(N + L)$ din același motiv ca mai sus.

Găsirea răspunsului

Există mai multe soluții de punctaje diverse pentru această parte a rezolvării.

Varianta ineficientă

Parcurgem numerele memorate în vectorul asociat cu primul element, generat la partea anterioară. Fie x un astfel de număr. Pentru a afla numărul de operații de a egaliza șirul la x, căutăm pe x în vectorii asociați cu restul elementelor din șir. Dacă x apare în **toți** vectorii, atunci însumăm numărul de operații și acela este răspunsul pentru a aduce toate elementele la valoarea x.

Acum repetăm procesul pentru orice x din primul vector, iar la final calculăm cel mai mic răspuns posibil.

Combinând și prima parte, această soluție are complexitatea timp aproximativ $O(N \cdot L^2)$. Soluția este prea înceată pentru punctaj maxim.

Varianta eficientă

Construim doi vectori de frecvență globali notați $câte$ și nr_ops. În $câte_x$ vom memora numărul de elemente din șir în care apare x în vectorul asociat. În nr_ops_x vom memora suma operațiilor de a transforma fiecare element din șir în x.

Pentru fiecare vector asociat cu un element din șir, este important ca mai întâi să îi eliminăm duplicatele pentru a putea actualiza cu ușurință cei doi vectori globali. O metodă de a face asta este să sortăm întâi fiecare vector și dacă avem mai multe valori identice alăturate vom lua în calcul **doar prima apariție** a valorii respective.

Parcurgem fiecare vector asociat. Fie x o valoare unică din vector; doar actualizăm vectorii globali astfel: câte_x crește cu 1 și nr_ops_x crește cu numărul de operații de a transforma valoarea inițială în x.

În final, parcurgem orice valoare posibilă $0 \leq x \leq 2 \cdot 10^5$ și calculăm minimul dintre nr_ops_x **doar** atunci când $\text{câte}_x = N$ (adică x apare în toți vectorii asociați).

Soluția finală are complexitatea timp $O(N \cdot L \cdot \log(L) + \text{vmax})$ și complexitatea de spațiu $O(N + \text{vmax})$, unde vmax este valoarea maximă din șir ($2 \cdot 10^5$). Soluția primește punctaj maxim.

Varianta și mai eficientă, alternativă

Observăm că soluția de mai sus depinde de variabila vmax și la timp dar și la memorie. Aceasta primește punctaj maxim datorită faptului că vmax $\leq 2 \cdot 10^5$ din restricții. Soluția următoare putea fi folosită în cazul în care vmax era foarte mare.

De la varianta anterioară păstrăm ideea de eliminare a duplicatelor din vectorii asociați, însă nu mai folosim cei doi vectori de frecvență globali. În schimb, pentru a găsi o valoare x care apare în toți vectorii asociați, putem calcula **intersecția** dintre aceștia.

Intersecția dintre doi vectori se poate face simplu cu o **interclasare**. Vectorii asociați sunt deja sortați și, mai mult, nu conțin elemente duplicate, deci nu vom avea probleme cu interclasarea.

Intersecția dintre toți vectorii se face treptat: întâi facem intersecția dintre primii doi vectori, copiem rezultatul într-un vector t, apoi intersecția dintre t și al treilea vector, copiem rezultatul în vectorul t, etc. La final vectorul t va conține intersecția tuturor vectorilor asociați, cu tot cu operațiile aferente fiecărei valori la care se poate egaliza șirul.

Complexitatea de timp este $O(N \cdot L \cdot \log(L))$ iar complexitatea de spațiu este $O(L)$, deoarece nu avem nevoie nici măcar de șirul inițial de numere.

Soluție completă 2, alternativă

Autor: Alin-Gabriel Răileanu
Soluția aceasta este relevantă deoarece complexitatea de timp este $O(N \cdot (\log(\text{vmax}) + \log(\text{N})))$, pe când cele de mai sus aveau complexități de cel puțin $O(N \cdot L)$, unde $L = \log^2(\text{vmax})$. În plus, memoria folosită de această soluție este $O(N)$.

Este important să înțelegem legătura dintre operațiile din enunț și reprezentarea numerelor în binar. Operația de tip 1 adaugă un bit de 1 la sfârșitul numărului, iar operația de tip 2 șterge ultimul bit.

Să considerăm cel mai lung prefix comun al tuturor numerelor (în baza doi). Observăm că valoarea la care egalizăm șirul trebuie să conțină neapărat acest prefix la început. Mai mult, se poate demonstra că, în valoarea la care egalizăm, **toți biții** de după prefixul comun **trebuie** să aibă valoarea 1.
Putem vizualiza mai ușor structura unui număr din șirul inițial. Acesta este format din trei părți "P A B", unde P este cel mai lung prefix comun al tuturor numerelor, A este cel mai lung șir format doar din 1, imediat după P, iar B este restul numărului.

Am spus că toți biții de după P trebuie să fie 1, deci e clar că din fiecare număr va trebui să renunțăm la B folosind operații de tip 2. Acum construim un nou șir în care valoarea unui element este egală cu lungimea lui A.

Mai departe putem reduce problema astfel:

> "Pe șirul nou construit avem două tipuri de operații: adună unu la un element (adică adăugăm un bit la partea A) și scade unu dintr-un element (adică scoatem un bit din partea A). Egalizează șirul folosind un număr minim de operații."

Aceasta este o problemă clasică și numărul minim de operații se obține egalizând șirul la **mediana șirului**.

Obținerea punctajului maxim pe această soluție nu necesită operații pe biți. În plus, dacă se folosesc counting sort pentru găsirea medianei și operații pe biți pentru găsirea prefixului comun, complexitatea de timp poate ajunge până la $O(N + \log(\text{vmax}))$.

Problema 3. Zapada

Propusă de: stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București

Cerința 1.

Pentru această cerință este necesar să găsim o strategie care, dat un prefix al șirului elimină în mod optim zăpada de pe acesta. Pentru sufix vom aplica aceeași strategie, dar de la început la final. Vom demonstra că este optim să mutăm zăpada de pe o dală exact o dată (există unele situații în care putem face două mutări, dar acestea sunt la fel de bune, deci nu ne vor interesa).

- Să presupunem că în soluția optimă folosim cel puțin lopețile k_1 și k_2 pe aceeași poziție, cu $k_1 \leq k_2$ (cazul $k_2 < k_1$ este echivalent).
- Putem muta $2^{k_1} + 2^{k_2}$ zăpadă cu costul $k_1 + k_2 + 2$.
- Dacă am folosi totuși lopata $k_2 + 1$, am putea muta 2^{k_2+1} zăpadă cu un cost de $k_2 + 2$.
- $2^{k_1} + 2^{k_2} \leq 2^{k_2} + 2^{k_2} = 2 \cdot 2^{k_2} = 2^{k_2+1}$. Deci cu lopetile k_1 și k_2 mutăm mai puțină sau la fel de multă zăpadă cât am muta cu lopata $k_2 + 1$.
- $k_1 + k_2 + 2 \geq k_2 + 2$. Deci cu lopețile k_1 și k_2 avem un cost egal sau mai mare decât dacă am folosi lopata $k_2 + 1$.
- Deoarece folosirea celor două lopeți nu ne aduce vreun beneficiu, este suficient să folosim doar o singură lopată.

Argumentul de deasupra ne spune că asupra fiecărei dale se va folosi o singură lopată. Pentru a muta zăpada de pe un prefix este deci optim să luăm dalele în ordine inversă și să găsim lopata minimă care poate muta toată zăpada de pe acea dală pe dala precedentă.

Acest raționament ne conduce la o soluție cu diverse implementări posibile. Orice soluție de complexitate $O(n \cdot \log Z)$ (Z este cantitatea totală de zăpadă) sau mai bună ar trebui să obțină punctaj maxim pentru această cerință.

Cerința 2.

Pentru această cerință este necesar să găsim poziția optimă de împărțire a șirului. În acest sens vom calcula eficient costul fiecărui punct de start și vom alege minimul dintre acestea. Vom calcula costul fiecărui prefix și apoi, separat, costul fiecărui sufix al șirului. Vom discuta

doar cum se calculează costurile prefixelor, deoarece costurile sufixelor se calculează în aceeași manieră.

Vom presupune că avem de calculat costul unui singur prefix de lungime L. Pentru a realiza acest calcul rapid vom face câteva observații.

Observație: Cantitatea de zăpadă ce va fi mutată de pe dala i pe dala $i-1$ este $z_i+z_{i+1}+\cdots+z_L$. Această sumă poate fi foarte ușor scrisă cu ajutorul sumelor parțiale. Fie s_i suma primelor i valori din z ($s_i = z_1 + z_2 + \cdots + z_i$). Atunci cantitatea de zăpadă mutată de pe dala i pe dala $i-1$ este $s_L - s_{i-1}$.

Ce putem face acum este în loc să calculăm pentru fiecare dală lopata necesară mutării zăpezii pe dala precedentă, putem calcula pentru fiecare lopată pe câte dale o folosim. Mai exact, lopata k va fi utilizată pentru mutarea zăpezii pe toate dalele i pentru care $2^{k-1} < s_L - s_{i-1} \leq 2^k$.

Pentru a afla rapid câte valori satisfac condiția de mai sus, o primă idee este să folosim căutarea binară. O asemenea soluție are complexitatea $O(n \cdot \log n \cdot \log Z)$ și ar trebui să se comporte destul de bine. Soluția de 100 de puncte folosește următoarea observație suplimentară.

Observație: Datorită faptului că trebuie să calculăm capetele intervalelor pentru fiecare prefix, nu este nevoie de căutarea binară. Putem în schimb să folosim tehnica Two Pointers.

Vom explica procesul pe următoarele două prefixe (a se observa că al doilea îl conține pe primul)

[5, 2, 3, 2, 6, 2, 1]

[5, 2, 3, 2, 6, 2, 1, 1]

Lopata 4 este folosită în primul prefix pentru dalele 2, 3, 4 și 5. În cel de-al doilea prefix lopata 4 este folosită pentru dalele 3, 4 și 5. În loc să căutăm binar din nou capetele intervalului, am putea să mutăm capetele pas cu pas.

Pentru fiecare lopată vom muta capătul stâng (analog pentru cel drept) al intervalului de cel mult n ori. Dacă am muta capătul stâng de mai mult de n ori, am ieși din vector ceea ce nu se poate. Fiecare din cele $\log Z$ lopeți face cel mult n pași deci complexitatea soluției este $O(n \cdot \log Z)$.

Echipa

Problemele pentru această etapă au fost pregătite de:

- prof. Emanuela Cerchez, Colegiul Național "Emil Racoviță" Iași
- șef lucrări dr. Mihai Nan, Facultatea de Automatică și Calculatoare, Universitatea Națională de Știință și Tehnologie Politehnica Bucureşti
- prof. Ionel-Vasile Piț-Rada, Colegiul Național Traian, Drobeta Turnu Severin
- prof. Sandor Lukacs, Colegiul Național "Onisifor Ghibu", Oradea
- prof. Filonela Bălașa, Colegiul Național "Grigore Moisil", București
- prof. Alina Gabriela Boca, Colegiul Național de Informatică Tudor Vianu, București
- prof. Vlad-Laurențiu Nicu, Liceul Teoretic Mihail Kogălniceanu, Vaslui

- stud. Eduard-Lucian Pîrțac, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași
- stud. Alin Gabriel Răileanu, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași
- stud. Matei Mocanu, Facultatea de Automatică și Calculatoare, Universitatea Tehnică "Gheorghe Asachi" Iaşi
- stud. Dumitru Ilie, Facultatea de Matematică-Informatică, Universitatea București
- stud. Mihnea-Alexandru Turcu, Facultatea de Matematică și Informatică, Universitatea "Babeș-Bolyai" Cluj
- stud. Cristian Luchian, Facultatea de Informatică, Universitatea "Al. I. Cuza" Iași
- stud. Daniel Gheorghe, Facultatea de Matematică-Informatică, Universitatea București