

Olimpiada Națională de Informatică,

Etapa națională, Clasa a x-a

Descrierea soluțiilor

Comisia științifică

March 27, 2026

Problema 1. BLIND

Propusă de: Prof. Ioan-Cristian Pop, Education Beyond Borders & Nerdvana 2.0, București

Cerința 1 (29 de puncte)

Ideea de rezolvare este să pornim cu căsuțele din exterior spre interior, adică să plecăm de la cele mai depărtate de punctul în care vrem să ajungem. Avem două soluții:

1. Folosim algoritmul lui Lee (plecând din (x_t, y_t)) pentru a determina distanța de la fiecare căsuță la destinație. Vom afișa apoi, toate căsuțele, în ordine descrescătoare a distanțelor. Căsuțele aflate la aceeași distanță pot fi afișate în orice ordine (două căsuțe aflate la aceeași distanță nu pot fi vecine)
2. Folosind algoritmul Fill (plecând din (x_t, y_t)), vom afișa o căsuță numai după ce am terminat de explorat toate căsuțele vecine.

Soluția generată astfel va fi de lungime fix K . Observăm că, nu este nevoie însă să afișăm căsuțele aflate la "margine" (căsuțele din care algoritmi Lee / Fill nu vor trece prin nicio căsuță vecină pornind din căsuța curentă). Exemplul din enunț prezintă această idee: nu este necesară afișarea căsuțelor $(1, 1)$ și $(2, 3)$.

Complexitatea soluției este $O(N \cdot M)$.

Cerința 2 (32 de puncte)

Plecăm de la soluția de la cerința 1: afișăm un set de instrucțiuni (cu lungime $< K$) astfel încât, indiferent de punctul de start, ne vom afla într-o căsuță (x, y) marcată cu 1 la final. De acolo, putem porni algoritmul Fill, doar că vom afișa de fiecare dată căsuța în care ne aflăm (când ajungem pentru prima oară și când ne întoarcem dintr-o căsuță vecină).

Lungimea setului de instrucțiuni este $2 \cdot K - 1$. Se poate observa că o parte dintre ultimele instrucțiuni afișate (cele care ne vor întoarce de unde am plecat, după ce am vizitat și ultima căsuță) nu trebuiesc afișate neapărat, dar nu este necesar pentru obținerea punctajului maxim.

Complexitatea soluției este $O(N \cdot M)$.

Cerința 3 (39 de puncte)

Vom porni cu K trasee în paralel, câte unul din fiecare căsuța marcată cu 1, și vom încerca să ținem minte pentru fiecare traseu, ce căsuțe am vizitat și ce căsuțe nu am vizitat încă. Atribuirea unui număr de ordin pentru fiecare căsuță este recomandată (transformare din coordonate (x, y) într-un număr de ordine z). Asociem fiecărui traseu un număr de ordin egal cu punctul de pornire, și marcăm punctul de pornire ca fiind vizitat. Pentru valori ridicate ale lui K , nu vom avea suficienta memorie să ținem o matrice de vizitare de tipul `int` sau `bool`. Vom avea nevoie de `std::vector<bool>` sau `std::bitset`.

De fiecare dată când citim o instrucțiune (x, y) , este posibil ca cel puțin două trasee să se unească (fiind plasate în căsuțe vecine). Vom încerca să unim traseele într-unul singur. Este redundant să simulăm mai multe trasee simultan care se află în aceeași căsuță. O soluție care nu unește trasee care se intersectează va obține în jur de 21 de puncte, în funcție de implementare.

Presupunem că, două trasee cu numerele de ordine z_1 și z_2 se intersectează atunci când se execută instrucțiunea (x, y) și ne dorim să continuăm doar cu traseul z_1 . Atunci, va trebui să actualizăm căsuțele vizitate – vor rămâne vizitate doar căsuțele vizitate de ambele trasee. Dacă traseul z_1 a vizitat căsuța c , dar traseul z_2 nu a vizitat căsuța c , atunci va fi nevoie să trecem din nou prin căsuța c (pentru că z_2 nu a trecut).

După ce am unit toate traseele în urma instrucțiunii (x, y) , vom marca căsuța (x, y) ca fiind vizitată de singurul traseu rămas. Atunci când un traseu a vizitat toate celelalte căsuțe, nu mai este nevoie să-l continuăm. Ne putem opri atunci când nu mai sunt trasee de continuat. Complexitatea soluției este $O(K^2 + Q)$ și va obține în jur de 28 de puncte.

Putem punctaj maxim, putem optimiza unirea a două trasee prin utilizarea unor structuri precum `std::bitset` – putem folosi operatorul `&` pentru a unii două trasee mult mai eficient decât prin parcurgerea căsuțelor una câte una. Complexitatea soluției rămâne aceeași, dar constanta este mult mai mică în dreptul lui K^2 . Se poate parsa și citirea, dar nu este necesară pentru obținerea punctajului maxim.

Problema 2. CALCULATOR

Propusă de: Stud. Alexandru Lorintz, Universitatea "Babeș-Bolyai", Cluj-Napoca

Soluția se bazează pe o observație fundamentală asupra modului în care procesele interacționează: dacă două sau mai multe procese au nevoie de cel puțin o variabilă comună, ele trebuie să ruleze secvențial. Astfel, timpul minim total necesar executării proceselor este dictat de „cea mai aglomerată” variabilă, adică acea variabilă pentru care suma timpilor proceselor care o accesează este maximă.

Fie un vector V , unde $V[x]$ reprezintă suma timpilor de execuție pentru toate procesele care accesează variabila x . Într-un scenariu optim de execuție, timpul global va fi exact valoarea maximă din acest vector.

Cerința 1

Pentru a construi eficient vectorul V , folosim metoda vectorului de diferențe (cunoscută în mediul olimpic și ca *șmenul lui Mars*). Pentru fiecare proces de la 1 la N , definit de intervalul $[l_i, r_i]$ și timpul t_i , vom face actualizările:

$$V[l_i] \leftarrow V[l_i] + t_i$$

$$V[r_i + 1] \leftarrow V[r_i + 1] - t_i$$

La final, calculând sumele parțiale pe vectorul V , vom obține timpul total pe stivă pentru fiecare variabilă. Răspunsul pentru prima cerință este $\max_{x=1}^M V[x]$, pe care îl vom numi `globalMax`.

Cerința 2

Pentru fiecare proces i eliminat, dorim să aflăm noul timp total T_i . Când procesul i dispare, valorile $V[x]$ pentru $x \in [l_i, r_i]$ vor scădea cu t_i , în timp ce valorile din afara intervalului vor rămâne neschimbate.

Noul maxim se poate obține fie dintr-o variabilă din exteriorul intervalului $[l_i, r_i]$, fie din interiorul lui. Pentru a afla rapid maximum din exterior, precalculăm doi vectori auxiliari: un prefix de maxime (`pref`) și un sufix de maxime (`suff`). Maximum din exterior, pe care îl vom nota cu `maxOut`, devine $\max(\text{pref}[l_i - 1], \text{suff}[r_i + 1])$. Pe de altă parte, maximum din interiorul intervalului nu poate depăși `globalMax` - t_i . Astfel, noul timp minim de execuție este:

$$T_i = \max(\text{maxOut}, \text{globalMax} - t_i)$$

Pentru cerința 2 se va afișa suma tuturor acestor valori T_i .

Cerința 3

Aici trebuie să determinăm numărul de variabile care ating noul timp maxim de execuție T_i (variabilele critice). Pentru a face asta eficient, în timpul precalculării prefixelor și sufixelor de maxime, vom menține și numărul de apariții ale acestor maxime (`prefMaxCnt` și `suffMaxCnt`), precum și un vector de sume parțiale pentru numărul de apariții ale valorii `globalMax` pe întreg vectorul (`prefCritical`).

Avem două cazuri principale când calculăm V_i :

1. $T_i = \text{globalMax}$. Asta înseamnă că eliminarea procesului i nu a redus deloc maximumul global. Astfel, variabilele critice sunt exact acele variabile care atingeau `globalMax` inițial și se află strict în **afara** intervalului $[l_i, r_i]$. Putem afla numărul lor scăzând din numărul total de variabile critice globale pe cele aflate în interval.

2. $T_i < \text{globalMax}$. Acum, noile variabile critice pot fi:

- Din interiorul intervalului: dacă $\text{globalMax} - t_i$ este exact noul maxim, atunci absolut **toate** variabilele din interior care atingeau vechiul maxim (care erau globalMax) vor atinge acum $\text{globalMax} - t_i$, devenind astfel critice. Adunăm la răspuns numărul lor total.
- Din exteriorul intervalului: dacă maxOut egalează noul maxim T_i , adunăm numărul de variabile din afara intervalului care ating valoarea maxOut , verificând pe rând din care parte provine maximul (din prefMaxCnt , din suffMaxCnt sau din ambele).

Complexitate timp și spațiu: Generarea datelor și algoritmul principal necesită o singură trecere sau un număr constant de treceri peste cele N procese și peste cele M variabile. Precalculările vectorilor de prefix și sufix necesită timp liniar în raport cu M . Astfel, complexitatea finală este de $O(N + M)$ timp și memorie.

Problema 3. CERCETAȘI

Propusă de: Stud. Matei Benchea, Universitatea București

Vom nota cu $S_r(n, k)$ numărul de moduri în care, dintr-o trupă cu n cercetași se pot forma exact k patrule știind că valoarea absolută a diferenței dintre indicii a doi cercetași ce se află în aceeași patrulă trebuie să fie cel puțin r . În alte cuvinte, problema ne cere să calculăm $\sum_{k=a}^b S_r(n, k)$ pentru n, r, a, b date.

Ne vom concentra acum pe a calcula $S_r(n, k)$. Vom încerca să derivăm o formulă de recurență. În primul rând, $S_r(0, 0) = 1$ indiferent de valoarea lui r . Să presupunem acum că am împărțit deja primii $n - 1$ cercetași în patrule și vrem acum să-l punem pe cercetașul cu numărul n . Distingem aici două posibilități:

1. Putem să-l plasăm pe n într-o patrulă nouă, lucru care ne este permis, căci nu ar exista un alt cercetaș în aceeași patrulă cu care n să aibă tensiuni. Numărul de moduri de a face acest lucru este $S_r(n - 1, k - 1)$.
2. Putem încerca să-l plasăm pe n într-o patrulă deja formată. Conform restricției date de r , n nu are voie să fie în patrulă cu niciunul dintre cercetașii: $n - 1, n - 2, \dots, n - r + 1$. Observația cheie care se face aici este că cercetașii $n - 1, n - 2, \dots, n - r + 1$ se află toți în patrule **diferite**. De ce este acest lucru adevărat? Considerăm cei mai îndepărtați doi cercetași de aici: $n - 1$ și $n - r + 1$ și vedem că $n - 1 - (n - r + 1) = r - 2 < r$, deci nu se pot afla în aceeași patrulă. Cum cei mai îndepărtați nu pot fi simultan în aceeași patrulă, conchidem că cercetașii $n - 1, n - 2, \dots, n - r + 1$ se află toți în patrule diferite. Asta înseamnă că din cele k patrule formate până acum, n poate intra în exact $k - r + 1$ patrule din cele k deja formate. Numărul de a face acest lucru este $(k - r + 1) \cdot S_r(n - 1, k)$

Ajungem astfel la următoarea relație de recurență:

$$S_r(n, k) = S_r(n - 1, k - 1) + (k - r + 1)S_r(n - 1, k)$$

Termenii acestei recurențe se pot calcula cu ajutorul unui algoritm de complexitate $O(n^2)$. Vrem totuși să îmbunătățim această soluție. În continuare vom dovedi că problema se reduce doar la calcularea recurențelor cu $r = 1$. Mai precis, vom arăta că $S_r(n, k) = S_1(n - r + 1, k - r + 1)$.

$$S_1(n - r + 1, k - r + 1) = S_1(n - r, k - r) + (k - r + 1)S_1(n - r, k - r + 1)$$

Aici facem următoarea notație: $G(n, k) = S_1(n - r + 1, k - r + 1)$. Rescriem relația de recurență și avem:

$$G(n, k) = G(n - 1, k - 1) + (k - r + 1)G(n - 1, k)$$

care se identifică exact cu relația de recurență pentru $S_r(n, k)$.

Așadar acum putem să ne concentrăm doar pe cazul cu $r = 1$. $S_1(n, k)$ reprezintă, de fapt, numărul de moduri în care putem să partiționăm o mulțime cu n elemente în k submulțimi. Conform principiului includerii și excluderii, acest număr de moduri se mai poate calcula și ca:

$$S_1(n, k) = \sum_{i=0}^k \frac{(-1)^{k-i} \cdot i^n}{(k-i)! \cdot i!}$$

care se poate calcula în $O(k \cdot \log(n))$, dacă folosim exponențiere rapidă. Revenind, problema ne cere să calculăm:

$$\sum_{k=a}^b \sum_{i=0}^k \frac{(-1)^{k-i} \cdot i^n}{(k-i)! \cdot i!}$$

care se poate face, fără vreo altă observație, în complexitate de $O((b-a) \cdot n \cdot \log(n))$.

Ca să ajugem la soluția optimă trebuie să încercăm să accelerăm procesul de calculare al sumelor. Acest lucru se poate face dacă schimbăm ordinea sumelor din formula anterioară:

$$\sum_{k=a}^b \sum_{i=0}^k \frac{(-1)^{k-i} \cdot i^n}{(k-i)! \cdot i!} = \sum_{i=0}^b \frac{i^n}{i!} \cdot \sum_{j=\max(0, a-i)}^{b-i} \frac{(-1)^j}{j!}$$

De ce ne ajută această scriere? Pentru suma interioară putem să ținem un vector de sume parțiale $pref_n = \sum_{i=0}^n \frac{(-1)^i}{i!}$ și putem să calculăm astfel suma interioară în $O(1)$. Această soluție are o complexitate de $O(n + b \cdot \log(n))$ sau $O(n \cdot \log(n) + b \cdot \log(n))$ în funcție de metoda de precalculare a factorialelor.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Cercetător științific Tamio-Vesa Nakajima, Universitatea Marburg, Hesse
- Prof. Mihai Bunget, Colegiul Național "Tudor Vladimirescu", Târgu Jiu
- Prof. Ioan-Cristian Pop, Education Beyond Borders & Nerdvana 2.0, București

- Stud. Alexandru Lorintz, Universitatea "Babeş-Bolyai", Cluj-Napoca
- Stud. Gheorghe Liviu Armand, Universitatea Bucureşti
- Programator Radu Muntean, TDB, Bucureşti
- Stud. Alexandru Peticaru, Universitatea Politehnică Bucureşti
- Stud. Tiberiu Cozma, Universitatea Alexandru Ioan Cuza, Iaşi
- Programator Răzvan Viorel Uzum, Google, Bucureşti
- Stud. Matei Benchea, Universitatea Bucureşti
- Stud. Cristian-Adrian Blaga, Universitatea "Babeş-Bolyai", Cluj-Napoca
- Stud. Laurian Iacob, Universitatea Bucureşti
- Stud. Raul Mihai Feier, Universitatea Politehnică Bucureşti