Olimpiada Națională de Informatică, Etapa Județeană
Clasa a X-a
Descrierea Soluțiilor

Comisia Științifică

Sâmbătă, 15 Martie, 2025

Problema 1: Golf

Propusă de: stud. Andrei Onuț, Yale University, Statele Unite ale Americii

Soluție pentru primul subtask. Pentru a determina numărul de celule din golf ce sunt umplute cu pământ, se pot citi caracterele (de `0` sau `1`) ce reprezintă elementele matricei A, unul câte unul. Așadar, într-o variabilă E, se poate reține răspunsul pentru prima cerință: astfel, pentru fiecare caracter de `1` întâlnit în A, se incrementează valoarea lui E. La final, după ce toate cele $n \times m$ elemente din matrice au fost citite, se afișează valoarea lui E. Complexitatea totală de timp este de $O(n \cdot m)$, iar spațiul utilizat se încadrează în $O(1)$ (nu este nevoie să stocăm niciun tablou/*vector* — totul se efectuează cu ajutorul unui număr mic, constant de variabile).

Soluție pentru al doilea subtask. De vreme ce se garantează (prin tabelul cu **Restricții și precizări** din enunț) că există o singură insulă în golf, înseamnă că numărul de insule (din Golful Biscayne) ce conțin un număr maxim de insule (adică răspunsul la cerința de rezolvat) este chiar 1.

Soluție pentru al treilea subtask. Pentru a determina numărul de celule ce fac parte din fiecare insulă a golfului, se poate folosi un algoritm ce adaugă, în timpul *descoperirii* unei insule, fiecare element de `1` într-o structură de date similară cu o *coadă* exact o singură dată (spre exemplu, folosind un tablou auxiliar în care se marchează vizitarea elementelor în cadrul unei iterații anterioare), cum ar fi *Algoritmul lui Lee*; dacă dorim să folosim o metodă recursivă, recomandăm și *Algoritmul de tip Flood Fill*.

Pentru fiecare insulă α, se poate stoca și o variabilă num_α (de tip `int`) care reține numărul de celule umplute cu pământ ce intră în alcătuirea sa. Apoi, printr-o parcurgere a acestei structuri de date (de exemplu, `struct`, în `C/C++`) ce reține informații despre insule (structura conține, în total, cel mult $O(n \times m)$ elemente, corespunzătoare insulelor), se calculează numărul elementelor num_α maximale.

Astfel, atât complexitatea de timp, cât și cea de spațiu, sunt de $O(n \times m)$.

Soluție pentru al patrulea subtask. Pentru fiecare interogare dintre cele Q, se pot găsi insulele ce *influențează* (altfel spus, modifică) rezultatul, într-o manieră similară cu cea pentru al treilea subtask. Astfel, să presupunem că avem de rezolvat o interogare de tipul (`C`, p) (unde $1 \leq p \leq m$). Fie $I_\alpha = \{(x_{\alpha,1}, y_{\alpha,1}), (x_{\alpha,2}, y_{\alpha,2}), \ldots, (x_{\alpha,k_\alpha}, y_{\alpha,k_\alpha})\}$ mulțimea ce conține **toate** cele k_α celule (reprezentate prin perechi formate din linia $x_{\alpha,i}$, respectiv coloana $y_{\alpha,i}$ — coordonatele celulelor în matricea A) ce intră în alcătuirea unei insule α; spunem că insula α are *dimensiunea* egală cu k_α.

Dacă $max(y_{\alpha,1}, y_{\alpha,2}, \ldots, y_{\alpha,k_\alpha}) < p$, înseamnă că insula α este situată la stânga coloanei p, așa că valoarea lui a (adică, conform enunțului, numărul de celule din toate insulele ce se află la stânga coloanei p, în cadrul interogării curente) crește cu k_α; altfel, dacă $min(y_{\alpha,1}, y_{\alpha,2}, \ldots, y_{\alpha,k_\alpha}) > p$, înseamnă că insula α este situată la dreapta coloanei p, așa că valoarea lui b (numărul de celule din toate insulele ce se află la dreapta coloanei p) crește cu k_α. În orice alt caz, valorile lui a și b nu se modifică (cu ocazia procesării insulei curente α).

În mod similar, putem raționa pentru o interogare de tipul (L, p) (unde $1 \leq p \leq n$) – în loc să folosim coordonatele de tip coloană $y_{\alpha,i}$, le vom folosi pe cele de tip linie $x_{\alpha,i}$.

Prin urmare, complexitatea totală de timp este de $O(Q \times n \times m)$, iar cea de spațiu este de $O(n \times m)$.

Soluție pentru al cincilea subtask. De vreme ce interogările de tip (L, p) pot fi rezolvate într-un mod analog cu cele de tip (C, p) (de exemplu, prin *înlocuirea* liniilor matricei cu coloanele acesteia și viceversa, sau prin folosirea unei structuri de date de forma `std ::pair <int, int>`, în C++), putem să presupunem, pentru simplicitate, că avem de răspuns doar la interogări de tipul (C, p).

Determinarea tuturor insulelor α (adică a coordonatelor celulelor ce intră în alcătuirea unei insule și a dimensiunii acesteia) se poate efectua în complexitatea (totală) de timp $O(n \times m)$, așa cum a fost descris anterior.

În cadrul unei insule α, să introducem următoarele două notații:

- $\mu_{y,\alpha} = max(y_{\alpha,1}, y_{\alpha,2}, \ldots, y_{\alpha,k_\alpha})$;
- $\omega_{y,\alpha} = min(y_{\alpha,1}, y_{\alpha,2}, \ldots, y_{\alpha,k_\alpha})$.

Mai mult, introducem și următoarele două tablouri unidimensionale $l[1, ..., m]$ și $r[1, ..., m]$, definite astfel (pentru fiecare i: $1 \leq i \leq m$):

$$l[i] = \sum_{\substack{\alpha \\ \mu_{y,\alpha}=i}} k_\alpha,$$

iar

$$r[i] = \sum_{\substack{\alpha \\ \omega_{y,\alpha}=i}} k_\alpha.$$

Cu alte cuvinte, $l[i]$ reprezintă suma dimensiunilor insulelor din golf ce au valoarea maximală a unei coloane (a unei celule din componență) egală cu i. Similar, $r[i]$ reprezintă suma dimensiunilor insulelor din golf ce au valoarea minimală a unei coloane egală cu i.

Așadar, răspunsul pentru o interogare de tipul (C, p) este dat de valoarea expresiei $a \times b$, unde:

$$a = \sum_{\substack{\alpha \\ \mu_{y,\alpha}<p}} k_\alpha = \sum_{\substack{\alpha \\ \mu_{y,\alpha}\in\{1,\ldots,(p-1)\}}} k_\alpha = \sum_{i=1}^{p-1} l[i]$$

și

$$b = \sum_{\substack{\alpha \\ \omega_{y,\alpha}>p}} k_\alpha = \sum_{\substack{\alpha \\ \omega_{y,\alpha}\in\{(p+1),\ldots,m\}}} k_\alpha = \sum_{i=p+1}^{m} r[i].$$

Atât a, cât și b, pot fi calculate printr-o parcurgere (liniară) a tablourilor $l[1, ..., m]$ și $r[1, ..., m]$. În funcție de elementul i de analizat (dacă $i < p$ sau $i > p$), se actualizează valoarea lui a sau valoarea lui b.

Astfel, complexitatea de timp necesară pentru a rezolva o interogare de tipul (C, p) este de $O(m)$. Similar, complexitatea de timp necesară pentru a rezolva o interogare de tipul (L, p) este de $O(n)$.

Prin urmare, complexitatea totală a acestui subtask este de $O(n \times m + Q \times max(n, m))$.

Soluție completă pentru $T = 3$. Aplicăm o metodă de rezolvare similară cu cea pentru al cincilea subtask. Observăm că, în cadrul unei interogări, pentru a determina, de exemplu, valoarea lui a, putem utiliza o tehnică de *sume parțiale*, pe prefixele șirului $l[1, ..., m]$; de asemenea, valoarea lui b poate fi calculată tot cu sume parțiale, pe sufixele șirului $r[1, ..., m]$.

Procesarea sumelor parțiale se poate efectua în complexitatea de timp (cât și de spațiu) de $O(m)$ (pentru coloane) sau de $O(n)$ (pentru linii).

Apoi, rezolvarea oricărei actualizări se poate efectua în $O(1)$.

Complexitatea de timp totală a acestei soluții este de $O(n \times m + Q)$. Spațiul utilizat se încadrează în $O(n \times m)$.

Problema 2: Bițșir

Propusă de: stud. Alexandru Dobleagă, Constructor University, Bremen, Germania

Subtask 1:

Când $X = 0$, șirul A_i trebuie să fie plin de 0 ($A_i = 0, \forall i \in \{1, 2, \ldots, N\}$). Fiindcă $Y = 0$, iar $M_i = 0, \forall i \in \{1, 2, \ldots, N\}$, răspunsul va fi mereu 1.

Subtask 2:

Când $X = 1$, șirul A_i trebuie să conțină cel puțin o valoare de 1, restul fiind egale cu 0. Din moment ce $Y = 0$, numărul de valori de 1 trebuie să fie par. Astfel, răspunsul va fi $C_2^N + C_4^N + \ldots = 2^{N-1} - 1$.

Subtask 3:

Observație: Șirul A_i nu poate avea elemente mai mari decât X.
Pentru acest subtask, se creează un for pentru fiecare valoare de la 0 la X și se verifică toate condițiile.

Subtask 4:

O generalizare a subtaskului 1, trebuie verificat dacă șirul plin de 0 respectă proprietățile **2** și **3**.

Subtask 5:

Pe baza observației anterioare și a mărimii șirului ($N \leq 4$), soluția pentru acest subtask este iterarea prin toate posibilitățile și verificarea celor 3 condiții.
Complexitate: $O(X^4)$ ca timp.

Subtask 6:

Observație: Operația de XOR are următoarea proprietate:

$$a \text{ XOR } b = c \Rightarrow a = b \text{ XOR } c$$

Soluție: Se iterează prin toate valorile posibile ale lui A_1 ($\{0, 1, \ldots X\}$), se află valoarea lui $A_2 = Y \text{ XOR } A_1$ și se verifică celelalte două condiții.
Complexitate: $O(X)$

Subtask 7:

În acest caz, cele 3 condiții pot fi reformulate astfel:

- Dacă $X = 1$, atunci există cel puțin o valoare de 1 în șir, altfel tot șirul este complet 0.
- Dacă $Y = 1$, atunci există un număr impar de 1 în șir, altfel există un număr par de 1.
- Dacă $M_i = 1$, atunci $A_i = 1$, altfel A_i poate fi 0 sau 1.

Soluție: Dacă $X = 0$, atunci problema se reduce la Subtaskul 4. Altfel, în funcție de valoarea lui Y și câte valori de 1 există în șirul M, se determină paritatea numărului de valori care pot fi alese. Mai exact, dacă se notează cu F numărul de valori de 1 din șirul M și cu R răspunsul, există următoarele cazuri:

- F impar și $Y = 0$: $R = C_1^{N-F} + C_3^{N-F} + \ldots = 2^{N-F-1}$

- F impar și $Y = 1$: $R = C_0^{N-F} + C_2^{N-F} + \ldots = 2^{N-F-1}$
- F par și $Y = 0$: $R = C_0^{N-F} + C_2^{N-F} + \ldots = 2^{N-F-1}$
- F par și $Y = 1$: $R = C_1^{N-F} + C_3^{N-F} + \ldots = 2^{N-F-1}$

CAZ SPECIAL:
Dacă $X = 1$, $F = 0$ și $Y = 0$, atunci la formulă se scade 1 deoarece se înnumără și șirul plin de 0.
Complexitate: $O(N)$

Subtask 8:

Din cauza naturii operațiilor pe biți, problema poate fi rezolvată bit cu bit, fiind împărțită astfel în 30 de subprobleme de tipul subtaskului 7. Mai exact, pentru bitul b, subproblema este următoarea:

- $X_b = 1$, dacă X conține bitul b, altfel $X_b = 0$.
- $Y_b = 1$, dacă Y conține bitul b, altfel $Y_b = 0$.
- $M_i = 1$, dacă M_i conține bitul b, altfel $M_b = 0$.

Se rezolvă toate subproblemele, iar răspunsul final va fi produsul tuturor răspunsurilor subproblemelor.

Problema 3: Anagrame

Propusă de: prof. Daniela Lica, Centrul Județean de Excelență Prahova

Subtask 1:

Complexitatea $O(M)$. Structura testelor de intrare face ca șirul suport să conțină aceeași literă ca șirul A, iar lungimea acestuia să fie minimul lungimilor intervalelor ce intervin în operațiile de generare, $min(y_q - x_q)$.

Subtask 2:

Complexitatea $O(N \times \Sigma \times M)$. Pentru fiecare literă L, situată în secvența interogată, determinăm $Range[L]$, în complexitate liniară $O(N)$, reținându-se prima și ultima apariție a ei în cadrul secvenței.

Subtask 3:

Complexitatea $O(N \times \Sigma + M)$. Întrucât nu se fac update-uri, se poate precalcula, pentru fiecare poziție din șir, pentru fiecare literă a alfabetului, $St[poz][lit]$ reprezentând cea mai apropiată poziție la stânga, respectiv $Dr[poz][lit]$ reprezentând ce mai apropiată poziție la drepta unde se regăsește aceasta. Complexitatea $O(N \times \Sigma)$. În urma precalculării răspunsul la o interogare se face în timp constant $O(1)$.

Subtask 4:

Complexitatea $O(M \times \Sigma \times log(N))$. Pentru fiecare literă se memorează, într-un set, pozițiile unde aceasta apare în șir. Prima și ultima poziție din cadrul intervalului interogat, pentru fiecare literă, se determină în $O(log(N))$ folosind căutarea binară. Șirul suport minim lexicografic poate fi determinat printr-o comparare pe vectori de frecvență.

Subtask 6:

Complexitatea $O(M \times (N \times \Sigma + (y - x)!))$. Lungimea secvenței interogate permite numărarea anagramelor șirului suport folosind *next_permutation* pentru generarea anagramelor distincte în ordine lexicografică.

Subtask 7:

Dacă șirul suport conține $l = c_a + c_b$ litere în total (adică, c_a litere de `a` și c_b litere de `b`), înseamnă că numărul de anagrame ale acestuia este egal cu $\frac{l!}{a! \times b!}$.

Subtask 8 (Generalizarea a subtask-ului 7):

Complexitatea $O(N \times \Sigma + M \times log(N) \times \Sigma)$. Numărul anagramelor distincte este egal cu numărul permutărilor cu repetiții. Presupunem că șirul suport are Nr caractere și conține litera l_1 de n_1 ori, litera l_2 de n_2 ori,....litera l_k de n_k ori, astfel încât $n_1 + n_2 + ... + n_k = Nr$. Numărul de permutărilor cu repetiții este egal cu $Nr!/(n_1! \times n_2! \ldots \times n_k!)$. Notăm cu $K = (n_1! * n_2! ... * n_k!)$ și cu $Mod = 1999999973$, atunci deoarece Mod este număr prim, $(Nr!/K) \equiv (Nr! \times K^{Mod-2})$ (modulo Mod). Se pot precalcula factorialele numerelor mai mici sau egale cu Nr iar exponențierea se implementează în complexitate logaritmică.

Echipa

Problemele pentru această etapă au fost pregătite de:

- Stud. Ariciu Toma, Universitatea Națională de Știință și Tehnologie POLITEHNICA București, Facultatea de Automatică și Calculatoare
- Stud. Dobleagă Alexandru, Constructor University, Bremen, Germania
- Prof. Lica Daniela, Centrul Județean de Excelență Prahova, Ploiești
- Instr. Măgureanu Livia, Universitatea București
- Drd. Nakajima Tamio-Vesa, Department of Computer Science, University of Oxford
- Stud. Onuț Andrei, Yale University, Statele Unite ale Americii
- Stud. Oprea Mihai-Adrian, ETH Zurich
- Prof. Pascu Olivia-Cătălina, Colegiul Național "Nichita Stănescu", Ploiești
- Prof. Pătcaș Csaba, Universitatea "Babeș-Bolyai", Cluj-Napoca
- Stud. Peticaru Alexandru, Universitatea Națională de Știință și Tehnologie POLITEHNICA București, Facultatea de Automatică și Calculatoare
- Stud. Verzotti Matteo-Alexandru, Universitatea București