Olimpiada Societății pentru Excelență și Performanță în Informatică, Etapa Națională
Descrierea Soluțiilor
Clasa a X-a

1 Problema Cvintete

Propunător: prof. Eugen Nodea

Cuvinte-cheie: divizori, frecvență de apariții, numărare, căutare binară, two/three pointers

Notații:

- $V = \max\{S[1], S[2], ..., S[N]\} = S[N]$ (valoarea maximă a elementelor din S)
- $nrDiv = \text{card}\{d \mid d \text{ divide } D, d \leq V\}$ (numărul de divizori mai mici sau egali decât V ai lui D)
- $nrTrip = \text{card}\{(a, b, c) \mid a \cdot b \cdot c = D\}$ (numărul de triplete care respectă condiția $a \cdot b \cdot c = D$)

Complexitățile soluțiilor le vom enunța conform cu notațiile de mai sus.

Subtask-ul 1 ($N \leq 20$)

Pentru acest subtask, putem aplica un algoritm tip "brute force", iterând prin toate cvintetele posibile de indici și verificând să se respecte condițiile din enunț.

Complexitatea acestei soluții este $O(N^5)$.

Subtask-ul 2 ($N \leq 1000$)

Pentru acest subtask, putem găsi toate tripletele (i_1, i_2, i_3) pentru care $S[i_1] \cdot S[i_2] \cdot S[i_3] = D$. Pentru fiecare triplet (a, b, c) descoperit prin acest proces, vom itera prin toate variantele (i_4, i_5), verificând pentru fiecare dacă $a \cdot S[i_4]^2 + b \cdot S[i_5]^2 = c^2$ și $S[i_4] \neq S[i_5]$. Dacă se respectă condiția, atunci vom adăuga la răspuns frecvența cu care apare tripletul (a, b, c).

Complexitatea acestei soluții este $O(N^3 + nrTrip \cdot N^2)$

Subtask-ul 3 ($N = V \leq 5000, S[i] = i$)

Pentru acest subtask, vom determina numărul de triplete în $O(nrDiv^2)$. Pentru asta, vom începe prin a ne precalcula divizorii lui D mai mici sau egali cu V în $O(V)$. Mai apoi, vom genera tripletele fie prin generarea perechilor (a, b) a.î. D se divide la $a \cdot b$ și alegerea $c = D/(a \cdot b)$, fie folosind trei pointeri, bazându-ne pe faptul că secvența

de divizori este ordonată:

```
T ← [ ]      // Lista de triplete
for k = nrDiv, 2, −1 do
    i ← 1, j ← k − 1
    while i < j do
        x ← div[i] · div[j] · div[k]
        if x = D then
            append (div[i], div[j], div[k]) to T
        end
    end
end
```

Pentru fiecare dintre aceste triplete (a, b, c), vom încerca toate variantele pentru x și vom calcula $z = c^2 - ax^2$. Dacă z este divizibil cu b, z/b este pătrat perfect și $z/b \neq x^2$, atunci vom incrementa răspunsul cu 1. Trebuie tratat cu atenție și cazul $z < 0$, în funcție de implementare.

Complexitatea unei astfel de soluții este $O(V + nrDiv^2 + nrTrip \cdot V)$.

Subtask-ul 4 ($N \leq 10\,000$, $V \leq 5\,000$)

Ca la subtask-ul anterior, generăm toate tripletele (a, b, c) cu produsul egal cu D. Pentru fiecare triplet (a, b, c) fixat și pentru o poziție i_4, setăm $x = S[i_4]$ și găsim (folosind căutarea binară pe șirul sortat S), intervalul de poziții posibile pentru i_5, astfel încât $ax^2 + by^2 = c^2$, unde $y = S[i_5]$.

Complexitatea unei posibile soluții este $O(V + nrDiv^2 + nrTrip \cdot N \log N)$.

Subtask-ul 5 ($N \leq 10^5$, $V \leq 20\,000$)

Pentru acest subtask, putem înlocui căutarea binară a lui y din soluția precedentă folosind vectori de frecvență. Mai exact, ne vom defini un vector ap cu semnificația ca $ap[i]$ este numărul de apariții a valorii i în șirul S. Astfel, putem obține toate cvintetele valide pentru un triplet (a, b, c) fixat cu o căutare cu doi pointeri, sau, mai simplu, folosind vectori de frecvență și formula $y = \sqrt{(c^2 - ax^2)/b}$.

Un posibil algoritm care găsește și contorizează toate cvintetele care se formează pentru un triplet (a, b, c) este următorul:

```
x ← 1, y ← V, nr ← 0
while x ≤ V and y ≥ 1 do
    r ← a · x^2 + b · y^2
    if r = c^2 and x ≠ y then
        nr ← nr + ap[x] · ap[y]
    end
    if r < c^2 then
        x ← x + 1
    else
        y ← y − 1
    end
end
```

O observație care reduce timpul de execuție simțitor este oprirea căutării cvintetelor pentru un triplet (a, b, c) atunci când $ax^2 > c^2$, echivalentă cu $x > c/\sqrt{a}$.

O altă observație este că nu toate tripletele au potențial să verifice relația $ax^2 + by^2 = c^2$, ci doar acele tripletele pentru care $\gcd(a, b)$ divide c^2. Această observație, însă, nu este necesară pentru a obține punctaj maxim.

Complexitatea acestei soluții este $O(N + nrDiv^2 + nrTrip \cdot V)$.

2 Problema Primar

Propunători: conf. dr. Doru Anastasiu Popescu, stud. Lucian Bicsi

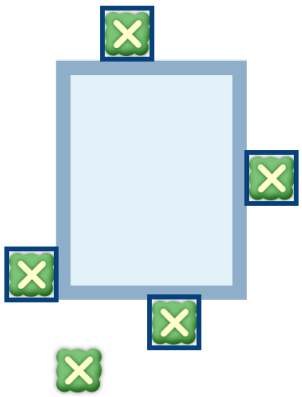
Cuvinte-cheie: ad-hoc, geometrie, sortări

Notații:

- $V = \max(\{x_i \mid 1 \leq i \leq N\} \bigcup \{y_i \mid 1 \leq i \leq N\})$ (valoarea maximă a unei coordonate din input)

Complexitățile soluțiilor le vom enunța conform cu notațiile de mai sus.

Vom începe prin a traduce condiția de *maximaliate* din enunț într-una echivalentă, dar cu care putem opera mai ușor:



Afirmație: *O regiune este maximală dacă și numai dacă este adiacentă la câte un arbore în fiecare dintre cele patru direcții (sus, jos, stânga, dreapta)*

Demonstrație: Într-adevăr, dacă o regiune este adiacentă câte unui arbore pe toate direcțiile, orice altă regiune care ar include-o (strict) ar conține măcar unul dintre cei patru arbori. Invers, să presupunem că o regiune nu conține arbori pe cele patru direcții (de exemplu, nu conține arbori în stânga). Atunci această regiune ar mai putea fi extinsă în acea direcție (stânga, în exemplul nostru) cu o unitate, obținând astfel o regiune care este în continuare validă, ceea ce ne determină să concluzionăm că regiunea inițială nu era maximală.

Notă: Observația de mai sus nu este strict necesară pentru a rezolva primele subtask-uri, însă folosirea ei face rezolvarea tuturor subtask-urilor considerabil mai ușoară.

Subtask-ul 1 ($N \leq 4$)

Pentru a rezolva acest subtask, începem prin a observa că pentru $N \leq 3$ nu există nicio regiune maximală (deci răspunsul este egal cu 0). Pentru cazul $N = 4$, poate exista cel mult o regiune maximală, determinată de bounding box-ul format de cele patru puncte $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$.

Mai exact, considerând:

- $x_l = \min\{x_1, x_2, x_3, x_4\}$
- $x_r = \max\{x_1, x_2, x_3, x_4\}$
- $y_d = \min\{y_1, y_2, y_3, y_4\}$
- $y_u = \max\{y_1, y_2, y_3, y_4\}$

răspunsul este dat de produsul $(x_r - x_l - 1)(y_u - y_d - 1)$, dacă toate cele patru puncte sunt pe marginile regiunii, sau 0 altfel.

Subtask-ul 2 ($N \leq 20$)

Pentru a rezolva acest subtask, vom încerca toate regiunile dreptunghiulare posibile $[x_l, x_r] \times [y_d, y_u]$ (eventual, considerând cei patru arbori de pe marginea acestora) și vom verifica dacă sunt valide.

Pentru a verifica dacă o regiune este validă, verificăm să existe pe fiecare margine măcar un arbore și să nu existe niciun arbore în interior. Condițiile pentru verificarea dacă un arbore (x, y) este (strict) în interior sunt:

$$(1)\ x_l < x < x_r \quad ; \quad (2)\ y_d < y < y_u$$

Complexitatea acestei soluții este $O(V^4 N)$ sau $O(N^5)$ (în funcție de implementare).

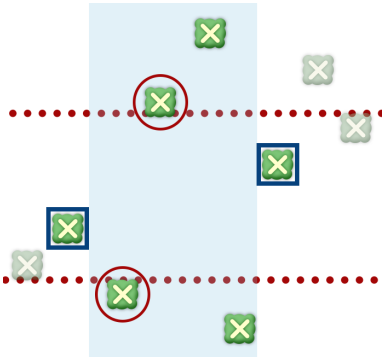
Subtask-ul 3 ($N \leq 120$)

Putem optimiza soluția anterioară, verificând condițiile necesare pentru ca o zonă să fie validă în timp $O(1)$ în loc de $O(N)$, folosind sume parțiale 2D (considerăm un arbore ca având valoare egală cu 1 și un spațiu liber ca având valoare egală cu 0).

Complexitatea acestei soluții este $O(V^4)$ sau $O(N^4 + V^2)$ (în funcție de implementare).

Subtask-ul 4 ($N \leq 500$)

Pentru a rezolva acest subtask, se observă că nu se mai poate itera prin toate regiunile dreptunghiulare posibile. Să presupunem însă că am fixat doi dintre cei patru arbori care definesc o regiune, mai exact cel din stânga și cel din dreapta.



Afirmație: *Pentru cei doi arbori fixați, există cel mult o regiune validă determinată de aceștia.*

Intuitiv, acest lucru se poate vedea în ilustrația din dreapta. Cei doi arbori fixați sunt marcați printr-un chenar albastru. Zona pe care aceștia o delimitează este marcată, iarăși, cu albastru. Cei doi arbori care delimitează latura de sus, respectiv de jos a unicii regiuni posibile sunt încercuiți cu roșu.

Demonstrația formală a afirmației de mai sus este lăsată ca exercițiu cititorilor.

O posibilă soluție este fixarea celor doi arbori (stânga/dreapta) și determinarea în $O(N)$ a celorlalți doi arbori (sus/jos), respectiv verificarea ca regiunea formată să fie valid definită și să nu conțină alți arbori în interior (similar cu soluțiile discutate mai sus).

Complexitatea acestei soluții este $O(N^3)$.

Subtask-ul 5 ($N \leq 1500$)

Putem optimiza soluția anterioară, sortând inițial arborii crescător după x și folosind structura de date `std::set` pentru a păstra arborii cu coordonatele $x_l < x < x_r$ în ordine sortată după y. Putem astfel găsi arborele de sus, reprezentând arborele cu cea mai mică coordonată y a.î. $y > \max\{y_l, y_r\}$, respectiv arborele de jos, reprezentând arborele cu cea mai mare coordonată y a.î. $y < \min\{y_l, y_r\}$, folosind metodele implementate `lower_bound`, `upper_bound`. Pentru a ne asigura că nu există arbori în această zonă, putem verifica dacă y_d și y_u sunt consecutivi în structura de date.

Complexitatea acestei soluții este $O(N^2 \log N)$.

Subtask-ul 6 ($N \leq 5000$)

Considerând soluția anterioară, putem face observația că merită să considerăm că latura de sus este determinată de arborele cu cea mai mare coordonată y a.î. $y > y_l$ și latura de jos este determinată de arborele cu cea mai mică coordonată y a.î. $y < y_l$. O regiune o vom lua în considerare doar dacă y_u, y_d sunt bine definite (ex. nu sunt valori infinite), iar $y_u > y_r$ și $y_d < y_r$.

Complexitatea acestei soluții este $O(N^2)$.

Notă: Există și soluții bazate pe stive (vezi problema determinării submatricei maxime conținând doar valori egale cu 0) care rezolvă problema în complexitate $O(N^2)$, dar care, în funcție de eficiența implementării în practică, pot obține sau nu punctajul asociat subtask-ului 6.

3 Problema Expresii

Propunători: stud. Tamio-Vesa Nakajima, stud. Lucian Bicsi.

Cuvinte-cheie: proprietăți gcd/lcm, descompunere în factori primi, evaluare de expresii

Notații:

- $V = \max\{b, \max(e)\}$ (valoarea maximă întâlnită în input)
- $E = |e|$ (lungimea expresiei)

Complexitățile soluțiilor le vom enunța conform cu notațiile de mai sus.

În redactarea acestei soluții **nu** vom folosi notațiile $(\cdot, \cdot)$ $[\cdot, \cdot]$ pentru a exprima noțiunile de cel mai mare divizor comun, respectiv cel mai mic multiplu comun. Vom alege, în schimb, notațiile explicite $\gcd(\cdot, \cdot)$, respectiv $\text{lcm}(\cdot, \cdot)$ (pentru claritate).

Subtask-ul 1 ($e =$ `(x,`t`)`)

În acest subtask, valoarea expresiei poate fi calculată ușor cu ajutorul algoritmului lui Euclid. Mai exact, pentru a afla soluția, calculăm $\gcd(x, t)$ pentru fiecare $x \in \{a, a+1, \ldots, b\}$, și însumăm. Pentru a afla valoarea lui t, este suficient să ignorăm primele 3 caractere, respectiv ultimul, din input (exemplu: "`(x,365)`"; caracterele colorate cu gri sunt ignorate).

Complexitatea acestei soluții este $O(E + V)$.

Subtask-ul 2 (fără `[, ]`)

Cum operația găsirii celui mai mare divizor comun este asociativă (adică: $\gcd(x, \gcd(y, z)) = \gcd(\gcd(x, y), z)$), putem deduce că, în acest caz, nu contează structura expresiei, ci doar constantele și valorile din aceasta. Astfel, pentru acest subtask, trebuie doar să eliminăm ignorăm parantezele și virgulele din expresie, să găsim constantele din expresie, și să vedem dacă "`x`" apare în expresie (exemplu: "`((x,18),(9,23))`"; caracterele colorate cu gri sunt ignorate).

Dacă constantele sunt $[c_1, c_2, \ldots, c_n]$, și notând cel mai mare divizor comun al tuturor acestor constante cu g, se disting două cazuri:

- dacă nu apare "`x`" în expresie, valoarea expresiei este g;
- dacă apare "`x`" în expresie, valoarea expresiei este $\gcd(g, x)$.

Aceste valori pot fi calculate ușor cu ajutorul algoritmului lui Euclid. Astfel, putem calcula valoarea expresiei pentru fiecare valoare a lui x de la a la b, și să le însumăm, ca în cazul precedent.

Complexitatea acestei soluții este $O(E + V)$.

Subtask-ul 3 ($E \leq 1\,000$, $V \leq 40$)

Pentru acest subtask, este suficient să evaluăm expresia pentru fiecare valoare x cu $a \leq x \leq b$. Deoarece orice rezultat parțial din expresie este mai mic sau egal cu $\text{lcm}\{1, 2, ..., 40\}$, este necesară folosirea tipurilor de date pe 64 de biți (`long long`).

Complexitatea acestei soluții este $O(E \cdot V)$.

Notă: Acesta este primul subtask pentru care este necesară parsarea efectivă a expresiei; de acum înainte vom presupune că acest pas a fost parcurs.

Subtask-ul 4 (constante puteri de 2)

Atenție! Acest subtask poate fi considerat mai dificil decât subtask-ul 5!

Orice număr natural x poate fi scris sub forma $2^k \cdot t$ pentru un t impar și un $k \geq 0$. Observăm că $\gcd(2^k \cdot t, 2^{k'}) = 2^{\min(k,k')}$, iar $\text{lcm}(2^k \cdot t, 2^{k'}) = 2^{\max(k,k')} \cdot t$. Acest lucru ne arată inductiv că dacă evaluăm expresia e pentru un $x = 2^k \cdot t$, indiferent de valoarea lui t, fie rezultatul va fi de forma $2^{k'}$ sau de forma $2^{k'} \cdot t$.

Pentru a evalua expresia într-un mod eficient, vom calcula $eval(e, 2^i \cdot t)$, pentru $0 \leq i \leq 16$. Trucul este următorul: **nu vom da o valoare specifică lui t!** În schimb, vom reprezenta numerele de forma $2^k \cdot t$ ca o pereche $(k, 1)$ și numerele de forma 2^k ca o pereche $(k, 0)$. Pentru a unifica cele două cazuri, vom spune că o pereche (k, l) reprezintă numărul $2^k \cdot t^l$ (cu $0 \leq k \leq 16$ și $l \in \{0, 1\}$).

Mai exact, să presupunem că x este de forma $2^i \cdot t$. Vom avea, de exemplu, că $eval(\text{“}\texttt{256}\text{”}) = 2^8 = (8, 0)$. De asemenea, $eval(\text{“}\texttt{x}\text{”}) = 2^i \cdot t = (i, 1)$.

Dar cum se pot generaliza aceste reprezentări? Să presupunem acum că e_1 și e_2 sunt expresii, iar $eval(e_1) = (k_1, l_1)$, $eval(e_2) = (k_2, l_2)$. Cât este atunci, $eval(\text{“}\texttt{(}e_1\texttt{,}e_2\texttt{)}\text{”})$? Ei bine, să ne amintim că $eval(e_1) = (k_1, l_1) = 2^{k_1} \cdot t^{l_1}$, iar $eval(e_2) = (k_2, l_2) = 2^{k_2} \cdot t^{l_2}$, ceea ce înseamnă că rezultatul pe care îl căutăm este exact $\gcd(2^{k_1} \cdot t^{l_1}, 2^{k_2} \cdot t^{l_2}) = 2^{\min(k_1,k_2)} \cdot t^{\min(l_1,l_2)}$, care este perechea $(\min(k_1, k_2), \min(l_1, l_2))$.

Analog putem demonstra că $eval(\text{“}\texttt{[}e_1\texttt{,}e_2\texttt{]}\text{”}) = (\max(k_1, k_2), \max(l_1, l_2))$ (recomandăm cititorilor ca exercițiu demonstrația).

Astfel, putem schița o soluție care să calculeze $eval(e, x)$ rapid:

- Calculăm rezultatele evaluărilor formale $(k_i, l_i) = eval(e, 2^i \cdot t)$ pentru $0 \leq i \leq 16$
- Descompunem $x = 2^q \cdot x'$
- Rezultatul este $eval(e, x) = 2^{k_q} \cdot (x')^{l_q}$

Complexitatea acestei soluții este $O(E \log V + V)$.

Notă: O soluție alternativă care rezolvă acest subtask constă în evaluarea expresiei pentru toate puterile de 2 și pentru un număr prim cu 2 (de exemplu, 3), și combinarea acestor rezultate pentru a obține o evaluare rapidă pentru un x oarecare.

Subtask-ul 5 ($E \leq 1000$, $V \leq 1000$)

Notă: Teoretic, soluția de la subtask-ul 3 ar avea toate șansele să obțină punctajul pentru acest subtask, însă, din păcate, în acest caz rezultatele intermediare (și finale) pot deveni mult prea mari (din cauza operației de lcm) pentru a putea fi reprezentate pe numere întregi pe 64 de biți.

Printr-o observație similară subtask-ului precedent, putem observa inductiv pe structura expresiei că $eval(e, x)$ acționează independent pe puterile factorilor primi ai lui x. Cu alte cuvinte, dacă presupunem că $x = p^q \cdot x'$, unde p este un număr prim, iar $\gcd(x', p) = 1$ atunci **exponentul lui p în rezultatul evaluării nu depinde de x'.**

Să presupunem acum că am (pre-)calculat $g(p, q)$ exponentul lui p în rezultatul evaluării $eval(e, p^q)$, pentru toate perechile (p, q) pentru care p este prim și p^q nu depășește $\max(e)$. Sunt cel mult $2 \cdot \max(e) = 2\,000$ astfel de perechi (în realitate, există o aproximare mai bună, dată de $O(\max(e)/\log(\max(e)))$).

Pe fiecare dintre numerele x cu $a \leq x \leq b$, îl vom descompune în factori primi sub forma:

$$x = p_1^{q_1} \cdot p_2^{q_2} \cdot \ldots \cdot p_n^{q_n}$$

unde $p_1, p_2, \ldots, p_n$ sunt numerele prime mai mici sau egale cu $1\,000$, iar $q_1, q_2, \ldots, q_n \geq 0$. Mai apoi, vom calcula $eval(e, x) = p_1^{g(p_1,q_1)} \cdot p_2^{g(p_2,q_2)} \cdot \ldots \cdot p_n^{g(p_n,q_n)}$, sumând apoi rezultatele. Produsele, respectiv suma, se vor efectua modulo $10^9 + 7$.

Complexitatea acestei soluții este $O(E \cdot V + V^2)$.

Subtask-ul 6 ($E \leq 250\,000$, $V \leq 10^5$, $a = b$)

Acest subtask cere, practic, evaluarea unei expresii foarte lungi, cu constante mult mai mari, dar într-o singură valoare. Bineînțeles, deoarece $a = b$, putem presupune $x = a$ și considera conceptual că expresia nu conține deloc variabila “$\texttt{x}$”. Cum putem evalua eficient o astfel de expresie?

Cheia e să folosim, în continuare, principiul descompunerii în factori primi. Fiecare număr și fiecare rezultat intermediar x îl vom reprezenta ca o mulțime $\{(p_1, q_1), ..., (p_n, q_n)\}$, cu convenția că $x = p_1^{q_1} \cdot p_2^{q_2} \cdot ... \cdot p_n^{q_n}$. Diferit de subtask-ul anterior, în acest caz vom menține doar exponenții diferiți de zero. Ca implementare, vom folosi structura de date `std::map` sau `std::unordered_map` pentru a reprezenta aceste mulțimi (pentru o soluție folosind structuri de date elementare, citiți soluția ultimului subtask).

Factorizare rapidă

În primul rând, merită de menționat că vom face o precalculare pentru a putea descompune rapid numerele în factori primi. Vom pre-calcula:

$$\text{small}(x) = \textit{ (cel mai mic / oricare) dintre divizorii primi al lui } x$$

Pre-calcularea aceasta se poate realiza în complexitate timp $O(V \log V)$ cu un algoritm similar ciurului lui Eratostene. Acest lucru ne permite să descompunem orice număr x în factori primi, împărțindu-l în mod repetat la $\text{small}(x)$ (până când acesta ajunge egal cu 1).

Evaluare rapidă

În această reprezentare, putem să ne amintim că gcd va alege exponentul minim pentru fiecare factor prim, pe când lcm va alege pe cel maxim.

Vom evalua expresia, returnând un rezultat ca o mulțime descrisă mai sus. Pentru cazul de bază (constantă), factorizăm numărul în timp logaritmic și returnăm mulțimea rezultată.

Pentru cazul lcm, combinăm cele două mulțimi, asigurându-ne că o introducem pe cea de dimensiune mai mică peste cea de dimensiune mai mare. Această tehnică ne permite să obținem o complexitate **amortizată** de $O(N \log N)$ (tehnică cunoscută și sub numele de "small to large merge"). Intuiția este bazată pe faptul că, de fiecare dată când iterăm printr-un element, acesta va fi introdus într-o mulțime de cel puțin două ori mai mare (sau va dispărea din universul de elemente).

Pentru cazul gcd, putem face un algoritm asemănător, sau chiar un algoritm care ne re-calculează intersecția în $O(N + M)$, dacă N și M sunt dimensiunile celor două mulțimi. Motivul este că, în cazul intersecției, numărul de elemente rezultat este mai mic sau egal decât $\min\{N, M\}$, ceea ce înseamnă că, în cadrul unei intersecții, cel puțin $\max\{N, M\}$ elemente "dispar" din univers. Acest lucru ne permite să executăm $O(\max\{N, M\}) = O(N + M)$ instrucțiuni fără a strica amortizarea.

Amortizat, această soluție are complexitate $O(F \log^2 F)$, unde F este numărul total de factori din cadrul constantelor din expresie. În teorie, am presupune că $F \in O(E \log V)$, dar în realitate, $F \in O(E)$, deoarece fiecare factor prim mai mare sau egal cu 10 crește lungimea expresiei cu măcar un caracter, deci nu pot fi mai mult de $O(E)$ factori în total.

Complexitatea unei astfel de soluții este între $O(E \log^2 E)$ și $O(E \log E)$, în funcție de implementare.

Subtask-ul 7 ($E \leq 250\,000$, $V \leq 10^5$)

Diferența între acest subtask și subtask-ul anterior este că acum trebuie să evaluăm expresia pentru x arbitrari. Vom prezenta două abordări diferite (care, însă, se dovedesc a fi strâns legate între ele).

V1. Reducere succesivă

Ideea acestei abordări se bazează pe reducerea expresiei e la una echivalentă, de forma:

$$R_{a,b}(x) = \text{lcm}(\text{gcd}(a, x), b)$$

pe care o vom reprezenta în implementare ca perechea (a, b), unde $a \geq 0, b \geq 1$. **În plus, punem condiția necesară ca a divizibil cu b.** Bineînțeles, a și b pot fi numere mari, deci vor fi reprezentate ca mulțimi de perechi (p, q), similar cu soluția de la subtask-ul anterior. Cazul $a = 0$ trebuie tratat explicit în implementare.

Însuși faptul că orice expresie poate fi rescrisă sub această formă (deși poate intuitiv), nu este trivial, iar demonstrația acestui lucru oferă și cheia descoperirii unui algoritm de rezolvare a problemei. Demonstrația se bazează pe proprietățile de distributivitate a gcd față de lcm (și invers), și rămâne ca exercițiu, cititorului.

În particular, se demonstrează că:

$$\gcd(R_{a,b}(x), R_{a',b'}(x)) = R_{\gcd(a,a'),\gcd(b,b')}(x)$$
$$\operatorname{lcm}(R_{a,b}(x), R_{a',b'}(x)) = R_{\operatorname{lcm}(a,a'),\operatorname{lcm}(b,b')}(x)$$

Notă: Este esențial, pentru corectitudinea formulelor, ca a să fie divizibil cu b. De ce? Se păstrează acest invariant în cadrul aplicării funcțiilor gcd, respectiv lcm?

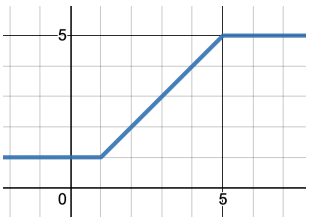
De asemenea, pentru cazurile de bază, avem că $R_{0,1}(x) = x$ și $R_{c,c}(x) = c$, pentru orice constantă c.

O dată forma aceasta definită, se transformă expresia e în forma $\operatorname{lcm}(\gcd(a, x), b)$, care se evaluează pe rând pentru fiecare x, în complexitate proporțională cu numărul de factori primi ai lui x. Ideea de bază este că pornim de la rezultatul b, la care înmulțim factorii primi p pentru care exponentul lui p în descompunerea lui $\gcd(a, x)$ este mai mare decât exponentul lui p în b.

V2. Exponenți ca funcții matematice

O altă variantă care se poate explora este ca rezultatul evaluării expresiei să codifice, pentru fiecare factor prim p, o funcție:

$$f_p(x) = \max(\min(a_p, x), b_p)$$



Aceste funcții au graficul similar cu ilustrația din dreapta (în ilustrația din dreapta, avem $a = 5, b = 1$. Intuitiv, evaluarea unei expresii într-un număr x se va considera independent pentru fiecare număr prim p, și va reprezenta produsul dintre $f_p(q)$, unde q este exponentul lui p în descompunerea în factori primi a lui x. Ca implementare, nu vom stoca în memorie funcțiile propriu-zise, însă vom ține perechile (a_p, b_p) pentru fiecare factor prim p.

Și aici, se vor distinge următoarele cazuri de bază: pentru $e = c$ (constantă), avem $a_p = b_p = q$, unde q este exponentul lui p în descompunerea în factori primi ai lui q, pentru $e = x$, avem $a_p = 0, b_p = \infty$, pentru orice p, iar pentru gcd și lcm, trebuie să calculăm funcțiile $\min(f_1, f_2)$, respectiv $max(f_1, f_2)$ (per element), care se traduc în a face $a = min(a_1, a_2)$ și $b = min(b_1, b_2)$.

Și în această variantă, trebuie tratat cu multă atenție cazul $a_p = 0, b_p = \infty$, pentru a nu strica complexitatea. Evaluarea expresiei pentru un x arbitrar se face similar cu cazul precedent.

Complexitatea ambelor soluții este $O(E \log^2 E + V \log^2 V)$ sau $O(E \log E + V \log^2 V)$, în funcție de implementare.

Bonus: *Care este legătura între cele două abordări prezentate mai sus?*

Subtask-ul 8 ($E \leq 2 \cdot 10^6$, $V \leq 250\,000$)

Pentru a rezolva acest ultim subtask, trebuie să optimizăm în continuare soluția anterioară.

În primul rând, operațiile de lcm se identifică cu operații de tip reuniune. Pe acestea le vom efectua în timp $O(1)$. Tehnica prin care vom realiza reuniunea în $O(1)$ este dată de faptul că nu vom realiza reuniunea (cel puțin nu în acel moment). În schimb, vom "concatena" listele de divizori. Reuniunea efectivă se va realiza în momentul în care apare o operație de gcd, în care vom efectua muncă proporțională cu $O(M + N)$ pentru a simula toate reuniunile și o intersecție (pentru care ne vom folosi de doi vectori globali, de capacitate $O(V)$).

Concatenarea rapidă a două liste se poate face în $O(1)$ folosind liste simplu înlănțuite, însă mai putem face o observație inteligentă: dacă ținem listele de perechi (p, q) serializate într-un buffer (stivă) global, reuniunea și intersecția a două liste se va face mereu între ultimele două liste, care sunt deja consecutive în buffer-ul global. Acest lucru implică faptul că reuniunea se poate face prin simpla modificarea unor indici ce indică pozițiile de început și final pentru liste, iar intersecția se poate face direct pe acea stivă, în timp $O(M + N)$. Astfel, complexitatea parsării expresiei în această formă devine $O(F) = O(E)$. *(egalitate argumentată la Subtask-ul 6)*

De asemenea, o altă optimizare mai puțin semnificativă se poate face după parsarea expresiei, prin popularea unor vectori caracteristici din listele de perechi, pentru a putea calcula rezultatul expresiei pentru un x dat în timp $O(\log x)$.

Complexitatea soluției finale este $O(E + V \log V)$.

Echipa

Setul de probleme pentru această rundă a fost propus și pregătit de:

- stud. doctorand Lucian Bicsi, Universitatea din București,
- prof. Mihai Bunget, Colegiul Naţional "Tudor Vladimirescu" Târgu Jiu,
- stud. Ștefan-Cosmin Dăscălescu, Universitatea din București,
- prof. Gheorghe Manolache, Colegiul Național de Informatică Piatra Neamț,
- stud. masterand Tamio-Vesa Nakajima, Universitatea Oxford,
- prof. Marius Nicoli, Colegiul Național "Frații Buzești" Craiova,
- prof. Eugen Nodea, Colegiul Naţional "Tudor Vladimirescu" Târgu Jiu,
- stud. Costin-Andrei Oncescu, Universitatea Oxford,
- stud. Bogdan-Petru Pop, Universitatea "Babeș-Bolyai" Cluj Napoca,
- conf.univ.dr. Doru Anastasiu Popescu, Universitatea din Pitești, Departamentul de Matematica-Informatica,
- stud. Mihai-Cristian Popescu, Universitatea Babeș-Bolyai Cluj Napoca.