E.I.L.I.M. (`eilim`)

Autor: Luca-Stefan Camburu

Contributor(i): Luca-Stefan Camburu

Soluție

Pentru a aborda această problemă, trebuie sp înțelegem ce fel de număr caută Luca. Deoarece nu are la dispoziție decât o singură întrebare, acesta trebuie să primească numărul Elisei în urma întrebării. Astfel spus, după notațiile din cerință, $\text{cmmdc}(X, Y) = Y$, deci Y este divizor al lui X.

Fie $p_1, p_2, p_3, \ldots, p_S$ numerele prime $\leq N$ (2, 3, ...) și $q_1, q_2, q_3, \ldots, q_S$ coeficienții cei mai mari posibili ai unui număr $\leq N$ (de exemplu, $p_1 = 2$, dacă $N = 5$, atunci $q_1 = 2$, pentru că $2^2 = 4 \leq N$, însă $2^3 = 8 > N$).

Dacă Elisa se gândește la orice număr de forma $p_i{}^{q_i}$, atunci X trebuie să fie multiplu de acest număr pentru a garanta că va fi găsit. Așadar, X trebuie să conțină toți factorii primi din intervalul $\{1, 2, \ldots, N\}$ la puterea cea mai mare posibilă. Deci numărul trebuie să fie divizibil cu cel mai mare multiplu comun al numerelor din $\{1, 2, 3, \ldots, N\}$. Îl dorim pe cel mai mic, așadar răspunsul la o întrebare este chiar $\text{cmmmc}(1, 2, 3, \ldots, N)$.

Problema se reduce acum la determinarea acestui număr, care e posibil să fie foarte mare, deci va fi nevoie de aritmetica numerelor mari.

Pentru $N \leq 40$, răspunsul este reprezentabil pe tipul de date `long long int`. Se pot parcurge toate numerele din $\{1, 2, \ldots, N\}$ și se actualizează treptat răspunsul, utilizându-se formula

$$\text{cmmdc}(a, b) \cdot \text{cmmmc}(a, b) = a \cdot b \Rightarrow \text{cmmmc}(a, b) = \frac{a \cdot b}{\text{cmmdc}(a, b)}.$$

Pentru calculul celui mai mare divizor comun, se va utiliza algoritmul lui Euclid. Această soluție rezolvă prima subproblemă, oferind 10 puncte.

Pentru $N, Q \leq 5000$, este nevoie de implementarea aritmeticii numerelor mari. Se poate utiliza același raționament ca la subproblema anterioară, însă calculele trebuie să fie rezolvate folosind operațiile specifice numerelor mari (înmulțirea unui număr mare cu un număr mic, împărțirea unui număr mare la un număr mic). Această soluție ofera încă 40 de puncte.

Pentru obținerea punctajului maxim, este nevoie de optimizări, dintre care menționăm (este de precizat că nu este nevoie de toate):

- reinterpretarea calculului: în loc de a face calculele pas cu pas, putem trata problema factor prim cu factor prim. Astfel, coeficientul fiecărui factor se va modifica doar la apariția numerelor $p, p^2, p^3, \ldots, p^q$, așadar nu este nevoie de a realiza o înmulțire la fiecare pas, ci doar la acești pași. Se reduce numărul de înmulțiri de la N, la cel mult 1280 (rezultat empiric). Este de precizat că va fi nevoie de utilizarea ciurului lui Eratostene pentru această soluție;
- optimizarea operațiilor cu numere mari: în loc de a utiliza calcule în baza 10, se poate alege o bază mai mare, astfel încât numărul de înmulțiri să fie cât mai mic. De exemplu, dacă alegem baza 10^9, atunci se reduce numărul de înmulțiri de 9 ori;
- reducerea memoriei: în loc de implementarea statică a numerelor mari, se poate utiliza container-ul STL `vector`. Marele avantaj este în copierea numerelor, deoarece se vor copia doar elementele care caracterizează numerele, nu și elementele "goale";

- optimizarea operațiilor de citire/afișare: deoarece volumul datelor este mare, se pot parsa intrarea și ieșirea datelor sau se poate utiliza decuplarea operatorilor `cin, cout` și desincronizarea de citirea C-style (`ios_base :: sync_with_stdio(0); cin.tie(0);`).

Șerpuit (`serpuit`)

Autor: Andrei-Cristian Ivan

Contributor(i): Andrei-Cristian Ivan

Soluție

Pentru subtaskul 1 ($N = 1$), matricea fiind formată dintr-un singur element, putem reține o variabilă în care vom stoca toate actualizările.

Pentru subtaskul 2 ($N = 2$), putem scrie toate cazurile de update posibile.

Pentru subtaskul 3 ($N \leq 2\ 000, K = 1$), stocăm toate actualizările într-o matrice.

Pentru subtaskul 4 ($K = 1$), ținem toate pozițiile actualizate într-un *map*.

Pentru subtaskul 5 ($N \leq 2\ 000, K = 2$), putem simula care ar fi următoarea poziție după prima în care se efectuează update-ul, iar update-urile le reținem într-o matrice.

Pentru subtaskul 6 ($N \leq 500, T \leq 500$), ne putem folosi de soluția de la subtaskul precedent pentru a parcurge matricea element cu element și a efectua actualizările. Complexitatea finală este $O(N^2 \cdot T)$.

Fiecare diagonală secundară are proprietatea că suma indicilor este constantă (vom nota $x + y = z$, unde (x, y) reprezintă o celulă din matrice).

Pentru subtaskul 7 ($N \leq 2\ 000, T \leq 2\ 000$), vom considera fiecare diagonală ca fiind un șir și vom itera prin toate diagonalele (începând cu diagonala z) pentru a face actualizările folosind șmenul lui Mars. Complexitatea finală este $O(N \cdot T + N^2)$.

Pentru subtaskul 8 ($N \leq 2\ 000$), o observație ilustrată chiar în enunțul problemei este că avem două tipuri de parcurgeri șerpuite: parcurgeri șerpuite care încep de pe diagonale cu z par și parcurgeri care încep de pe diagonale cu z impar. În loc să privim update-urile ca parcurgeri de diagonale într-o matrice, ne este mult mai ușor să le privim ca update-uri pe intervalele a două șiruri. Deci, vom liniariza cele două parcurgeri posibile și, pe cele două șiruri, vom face actualizări folosind șmenul lui Mars. La final, calculăm pentru fiecare celulă în parte cât este valoarea ei finală. Complexitatea finală este $O(N^2)$.

Pentru subtaskul 9, în loc să ținem în memorie toată matricea (care poate să conțină chiar și elemente care nu sunt importante pentru testul nostru), vom reține doar ce este relevant (poziții de început/final pentru actualizări și pozițiile de *query*). Vom aplica tehnica de evenimente, unde fiecare actualizare reprezintă câte două evenimente (în liniarizare, la poziția l avem evenimentul $+val$, iar la poziția $r + 1$ avem evenimentul $-val$), iar un *query* reprezintă alte două evenimente (câte unul în fiecare liniarizare în parte). Pentru un *query*, răspunsul final va fi suma răspunsurilor din cele două liniarizări. Complexitatea finală este $O((T + Q) \log(T + Q))$.