Despre două PROBLEME de la ONI 2002: noi SOLUȚII și EXTINDERI

Mihai Pătrașcu

Acest articol se referă pe scurt la două probleme propuse de autor la Olimpiada Națională de Informatică 2002. Enunțurile acestor probleme au fost publicate în numărul 12/5 (mai 2002) al GInfo. Articolul aduce elemente de noutate, prezentând câte o extindere și o nouă metodă de rezolvare pentru problemele Suma divizorilor și EQS.

Suma divizorilor

În [2] este exprimată opinia că rezolvarea problemei ***Suma divizorilor*** de la ediția 2002 a *Olimpiadei Naționale de Informatică* necesită unele elemente de matematică superioară. Luând în considerare rezolvarea publicată în [1] nu putem decât să fim de acord. Această rezolvare se baza pe existența inversului multiplicativ pentru orice număr nenul într-un corp de resturi modulo `9901` (care este un număr prim). Acest element de matematică ar trebui să fie accesibil elevilor de clasa a XII-a, însă nu și celor de clasa a XI-a (problema a fost propusă la clasele a XI-a și a XII-a).

Când a fost propusă această problemă, a fost luată în vedere și o soluție accesibilă elevilor de clasa a XI-a, care, pe lângă accesibilitate, mai are meritul că nu folosește faptul că `9901` este număr prim (astfel, problema este generalizată pentru numere care sunt resturi modulo numere compuse).

Pentru a prezenta soluția alternativă a acestei probleme, să ne reamintim că determinarea sumei divizorilor numărului A^B a fost redusă la calculul sumei unei progresii geometrice modulo `9901`: $S_n = (1 + q + q^2 + ... + q^n) \bmod 9901$ (vezi [1]).

Această sumă se poate calcula eficient dacă observăm următoarea recurență (toate calculele se efectuează modulo `9901`):

$$\begin{aligned} S_{2k+1} &= 1 + q + q^2 + ... + q^{2k} + q^{2k+1} = \\ &= 1 + q \cdot (1 + q + q^2 + ... + q^{2k}) = \\ &= 1 + q \cdot S_{2k}. \end{aligned}$$

$$\begin{aligned} S_{2k} &= 1 + q + q^2 + ... + q^k + q^{k+1} + ... + q^{2k} = \\ &= (1 + q + q^2 + ... + q^k) + q^k \cdot (q + q^2 + ... + q^k) = \\ &= S_k + q^k \cdot (S_k - 1). \end{aligned}$$

Dacă folosim o metodă cu timp de execuție logaritmic pentru a calcula valoarea q^k se observă că suma poate fi evaluată într-un timp cu ordinul de complexitate $O(\log^2 N)$. O astfel de metodă pentru calculul valorilor q^k este folosirea unei recurențe similare cu cea de mai sus:

$$\begin{aligned} q^{2k+1} &= q \cdot q^{2k} \\ q^{2k} &= (q^k)^2. \end{aligned}$$

Deși folosirea acestui algoritm cu ordinul de complexitate $O(\log^2 N)$ era suficientă pentru a obține punctajul maxim la concurs, se observă că algoritmul poate fi îmbunătățit pentru a rula într-un timp de ordinul $O(\log N)$ dacă îmbinăm cele două recurențe și calculăm simultan q^n și S_n:

$$\begin{aligned} q^{2k+1} &= q \cdot q^{2k} & \qquad S_{2k+1} &= 1 + q \cdot S_{2k} \\ q^{2k} &= (q^k)^2 & \qquad S_{2k} &= S_k + q^k \cdot (S_k - 1). \end{aligned}$$

Înainte de a încheia discuția problemei, dorim să combatem o obiecție pe care am auzit-o referitor la problemă, și anume că unii concurenți puteau să nu cunoască unele proprietăți cum ar fi:

$(A + B) \bmod C = ((A \bmod C) + (B \bmod C)) \bmod C.$

Considerăm că o astfel de obiecție este total nejustificată, având în vedere nivelul înalt de pregătire pe care ar tre-

bui să îl aibă participanții la faza națională a olimpiadei. În primul rând, astfel de proprietăți sunt intuitive și se folosesc încă din clasa a V-a când se calculează ultima cifră a rezultatului unor expresii. În al doilea rând, calculul cu resturi este destul de important în informatică (numere pseudoaleatoare, permutări, *hashing* etc.), astfel încât orice elev cu un interes deosebit pentru informatică (cum ar trebui să fie concurenții de la *ONI*) să îl fi cunoscut și folosit.

EQS

Așa cum a fost propusă, problema ***EQS*** cere numărarea soluțiilor dintr-un anumit domeniu de forma $[-A, +A]$ a unor ecuații de tipul :

$$a_1 \cdot x^3 + a_2 \cdot y^3 + a_3 \cdot z^3 + a_4 \cdot u^3 + a_5 \cdot v^3 = 0.$$

Pe scurt, soluția consta în a trece două dintre necunoscute în membrul drept al ecuației, schimbând semnul pentru coeficienți. Se generau apoi toate valorile posibile pentru membrul cu două necunoscute și se sortau. În continuare, se generau toate valorile posibile pentru membrul cu trei necunoscute și se căutau logaritmic între valorile posibile ale celuilalt membru al ecuației, folosind o căutare binară. Așadar, se folosește o cantitate de memorie de ordinul $O(A^2)$ (pentru tabloul sortat) și este necesar un timp de execuție de ordinul $O(A^3 \cdot \log A)$ (factorul dominant este căutarea în tabloul sortat).

Propunem în continuare o extensie a acestei probleme pentru ecuații cu șase necunoscute de forma:

$$a_1 \cdot x^3 + a_2 \cdot y^3 + a_3 \cdot z^3 + a_4 \cdot u^3 + a_5 \cdot v^3 + a_6 \cdot v^3 = 0.$$

Menționăm că această extensie s-a aflat în vederea comisiei de la *ONI*, dar nu a fost folosită în concurs, rezolvarea fiind considerată destul de dificilă.

Încercând să folosim soluția de mai sus pentru această nouă problemă, ne lovim de imposibilitatea de a păstra în memorie toate valorile posibile ale unui membru cu trei necunoscute.

Numărul acestor valori poate atinge $100^3 = 1.000.000$; astfel, ar fi necesară o cantitate de memorie disponibilă de 4MB, ceea ce nu a fost cazul la *ONI* 2002.

Intuim fără mare dificultate soluția: dacă am putea să generăm toate valorile posibile ale unui membru cu trei necunoscute în ordine crescătoare, am putea interclasa pur și simplu listele de valori posibile pentru membrul stâng și membrul drept.

Algoritmul ar necesita un spațiu de memorie constant și un timp de execuție constant în plus față de codul de generare a valorilor.

Ca urmare, subproblema care trebuie rezolvată constă în generarea în ordine crescătoare a elementelor următoarei mulțimi:

$$S = \{a \cdot x^3 + b \cdot y^3 + c \cdot z^3 \mid x, y, z \in [-A, A]\}.$$

În primul rând se observă faptul că semnul coeficienților nu contează, deci putem lucra numai cu coeficienți pozitivi.

Într-adevăr, următoarea egalitate este valabilă:

$$\{a \cdot x^3 + b \cdot y^3 + c \cdot z^3 \mid x, y, z \in [-A, A]\} = \{|a| \cdot x^3 + |b| \cdot y^3 + |c| \cdot z^3 \mid x, y, z \in [-A, A]\}.$$

Egalitatea acestor mulțimi se datorează faptului că, dacă este luat în considerare un triplet de forma (x, y, z), atunci vor fi luate în considerare toate tripletele de forma $(\pm x, \pm y, \pm z)$. În cazul în care coeficienții sunt pozitivi, există o relație de ordine parțială foarte utilă între valorile mulțimii și anume:

$$z_1 < z_2 \Rightarrow a \cdot x^3 + b \cdot y^3 + c \cdot z_1^3 \le a \cdot x^3 + b \cdot y^3 + c \cdot z_2^3.$$

Soluția pe care o propunem se bazează pe această relație de ordine parțială. Vom păstra un tablou bidimensional `t` de dimensiuni $2 \cdot A \times 2 \cdot A$ în care vom păstra, pentru fiecare pereche (x, y), cea mai mică valoare z pentru care tripletul (x, y, z) nu a fost încă luat în considerare. Versiunea în pseudocod a algoritmului este următoarea:

se inițializează tabloul `t` *astfel încât* $t_{ij} = -A$
cât timp *există elemente neutilizate ale mulțimii* **execută**
  găsește o pereche (x, y) *care minimizează valoarea expresiei* $a \cdot x^3 + b \cdot y^3 + c \cdot t_{xy}^3$ (*)
  următorul element al mulțimii este $a \cdot x^3 + b \cdot y^3 + c \cdot t_{xy}^3$
  $t_{xy} \leftarrow t_{xy} + 1$
sfârșit cât timp

Ordinul de complexitate al operației de generare a unui element este dictat de eficiența cu care putem efectua operația marcată cu `(*)` în algoritmul descris anterior. Pentru a executa eficient această operație, vom menține un *heap* în care vor fi păstrate toate perechile (x, y). Acest *heap* va implementa o coadă de priorități, unde prioritatea unui element (x, y) este, bineînțeles, chiar $a \cdot x^3 + b \cdot y^3 + c \cdot t_{xy}^3$.

Perechea (x, y) "minimă" va fi regăsită în timp constant (este chiar vârful *heap*-ului), iar pentru a menține structura *heap*-ului va mai fi necesar un timp de ordinul $O(\log A^2) = O(2 \cdot \log A) = O(\log A)$, deoarece prioritatea perechii (x, y) se schimbă în momentul incrementării valorii t_{xy}, deci elementul va trebui "cernut" în *heap*.

În concluzie, algoritmul rulează în timp $O(A^3 \cdot \log A)$, și are nevoie de un spațiu de memorie de ordinul $O(A^2)$. În mod suprinzător, aceste limite asimptotice sunt identice cu cele pe care le atingea soluția mai puțin performantă în cazul problemei "ușoare" cu cinci necunoscute.

Bibliografie

1. Mihai Scorțaru, *Olimpiada Națională de Informatică*, ***GInfo*** 12/7 (noiembrie 2002), p. 8-13, Editura Agora Media, Târgu Mureș
2. Mihai Scorțaru, *După ONI* 2002, ***GInfo*** 12/8 (decembrie 2002), p. 40-43, Editura Agora Media, Târgu Mureș

Mihai Pătrașcu este student la Massaschussets Institute of Technology (MIT) din Boston (Statele Unite ale Americii) și poate fi contactat prin e-mail la adresa **mip@mit.edu**.