Descrierea Soluțiilor
Olimpiada Societății pentru Excelență și Performanță în Informatică, Etapa Națională
Baraj Seniori 2

1 Problema Pastile

Propunător: Bogdan Sitaru; Bogdan-Petru Pop

Observația principală a problemei este că vrem să numărăm câte șiruri a_i există, astfel încât pentru fiecare $1 \leq i \leq N$, se respectă $1 \leq a_i \leq p_i$ și $\gcd(a_1, \ldots, a_N) = 1$. Astfel, fiecare nuanță (clasă de echivalență) va fi reprezentată în numărare de șirul care conține cel mai mic număr total de pastile.

Subtask 1. Se poate observa că există cel puțin un $p_i = 1$, deci oricum am alege $1 \leq a_1 \leq p_i$, cel mai mare divizor comun va fi mereu 1. Soluția este să calculăm produsul $\Pi_{i=1}^{N} p_i$.

Subtask 2, 3. Vom genera toate șirurile posibile în $O((V_{max})^N)$, iar pentru fiecare, vom verifica în $O(N)$ dacă $\gcd(a_1, \ldots, a_N) = 1$. Pentru a lua și subtaskul 3, în timp ce generăm trebuie să ținem în paralel și cel mai mare divizor comun al numerelor de pâna acum din șir, reducând complexitatea totală la $O((V_{max})^N)$.

Subtask 4. Vom rezolva problema în complexitate $O(N \cdot (V_{max})^2 \cdot \log(V_{max}))$ prin programare dinamică, calculând $dp_{i,j}$ - numărul de șiruri formate din pastile din primele i nuanțe inițiale care au $\gcd(a_1, \ldots, a_i) = j$. Vom calcula incrementând i-ul de la 1 la N. Astfel, putem selecta $d = \gcd(a_1, \ldots, a_{i-1})$ și v (ambele între 1 și V_{max}), și să adunăm $dp_{i-1,d}$ la $dp_{i,\gcd(d,v)}$.

Subtask 5. Vom folosi definiția precedentă pentru $dp_{i,j}$, însă o vom calcula în complexitate $O(N \cdot V_{max} \cdot \log(V_{max}))$. Pentru a calcula $dp_{i,j}$, vom calcula mai întâi $dp2_{i,j}$ - numărul de șiruri din primele i care au cel mai mare divizor comun j sau un multiplu al lui j. Astfel, $dp2_{i,j} = \lfloor \frac{p_i}{j} \rfloor \cdot \sum_{k \text{ multiplu de } j} dp_{i-1,k}$. După aceasta, putem calcula $dp_{i,j}$ folosind $dp2i, k$, cu k multiplu de j, cu ajutorul principiului includerii și excluderii.

Observații Pentru rezolvarea următoarelor subtaskuri vom proceda în următorul fel: fie cnt_i - numărul de șiruri a_i care sunt în limitele date și $\gcd(a_1, \ldots, a_N)$ este un multiplu al lui i. Observăm că, folosind principiul includerii și excluderii, putem calcula câte șiruri au $\gcd(a_1, \ldots, a_N) = 1$. Mai exact, acest număr este $cnt_1 - \sum_{k=2}^{(V_{max})} cnt_k * möbius(k)$. Există și alte metode de calcul ale răspunsului final folosind șirul cnt, care se bazează în mod asemănător cu soluțiile precedente

Putem observa că, de fapt, $cnt_i = \Pi_{j=1}^{N} \lfloor \frac{p_j}{i} \rfloor$. Tot ce ne rămâne este să calculăm eficient cnt_i.

Subtask 6. Pentru subtaskul 6, cnt_i se poate calcula în complexitatea $O(N^2)$, parcurgând pentru fiecare i toate j-urile de la 1 la N.

Subtask 7. Pentru subtaskul 7, cnt_i se poate calcula în complexitatea $O(N \cdot \sqrt{V_{max}} \cdot \log V_{max})$ folosind obervația că, pentru oricare $a \in \mathbb{N}$,

$$\left| \left\{ \left\lceil \frac{a}{x} \right\rceil \mid x \in \mathbb{N} \right\} \right| = O(\sqrt{a}).$$

Subtask 8. Aici $V_{min} = V_{max}$, deci toate valorile sunt egale (cu p_1, de exemplu). Astfel, $cnt_i = (\lfloor \frac{p_1}{i} \rfloor)^N$, care poate fi calculat în $O(\log(N))$ pentru fiecare i, rezultând într-o complexitate de $O(V_{max} \cdot \log(N))$ pentru calcularea șirului cnt_i.

Subtask 9. Aici $V_{min} \leq 100$. Se poate observa că $\gcd(a_1, \ldots, a_N) \leq V_{min}$, deci ne trebuiesc doar elementele cnt_i cu $i \leq V_{min}$. Astfel, putem folosi soluția de la subtaskul 6 pentru a calcula doar primele V_{min} elemente, rezultând complexitatea de $O(N \cdot V_{min})$ pentru calcularea șirului cnt.

Subtask 10. Pentru ultimul subtask (și pentru a rezolva problema în totalitate), vom calcula cnt_i în complexitatea $O(V_{max} \cdot \log(V_{max}) \cdot \log(N))$. Astfel, se poate observa că pentru fiecare valoare i, există $\lceil \frac{V_{max}}{i} \rceil$ valori diferite în calculul lui cnt_i. Prin sume parțiale, se poate calcula pentru fiecare valoare j de la 1 la $\lceil \frac{V_{max}}{i} \rceil$ exponentul la care acesta este în produsul pentru cnt_i, iar după produsul pentru cnt_i se va calcula prin exponențiere rapidă în complexitate $O(\lceil \frac{V_{max}}{i} \rceil \cdot \log(N))$. Însumând complexitatea pentru fiecare i de la 1 la V_{max}, rezultă complexitatea finală de $O(V_{max} \cdot \log(V_{max}) \cdot \log(N))$.

2 Problema Pswap

Propunător: Bogdan-Petru Pop; co-pregătitor: Tamio-Vesa Nakajima

Subtask 1. Pentru primul subtask, este suficient să creem graful relațiilor de similaritate și să generăm toate submulțimile posibile, și să o selctăm pe cea cu grad maxim. Complexitatea este astfel $O(N^2M + N2^N)$.

Subtask 2. Pentru cel de-al doilea subtask, trebuie să găsim o metodă mai eficientă de găsire a grafului. Observăm că douaă permutări sunt similare dacă diferă în exact 2 poziții. Putem verifica mai eficient acest lucru găsind cel mai lung prefix comun (fie acesta subcevența $(0, l)$, respectiv cel mai lung sufix comun (fie acesta subsecvența $(r, M-1)$. Putem calcula aceste 2 valori în $O(\log M)$ folosind hashing și căutare binară. Dacă știm valorile l și r, ajunge să folosim hashing pentru a determina dacă și secvența $(l+1, r-1)$ este identică în două permutări. Astfel, obținem o complexitate de $O(N^2 \log M)$ pentru construcția grafului, iar apoi putem aplica același algoritm de generare al mulțimilor (observați că maxim 20 de permutări sunt relevante pentru acest algoritm în acest subtask).

Subtask 3. Pentru cel de-al treilea subtask, putem construi graful ineficient, dar trebuie să găsim o metodă de a găsi rapid mulțimea maximă independentă a grafului de similarități. Observația principală este că o operație swap schimbă paritatea unei permutări, prin urmare graful este bipartit (partiția permutărilor după paritate împarte graful în cele două părți ale grafului). Aplicând Teorema lui König, putem demonstra că acest răspunsul este egal cu $N - c$, unde c este mărimea cuplajului maxim în graf. Putem demonstra că numărul de muchii ale grafului este de ordinull $O(N\sqrt{N})$ (pornind de la observația că 2 permutări au cel mult 3 permutări similare comune), deci în funcție de algoritmul de cuplaj utilizat putem obține complexitatea $O(N^2\sqrt{N})$ sau $O(N^2)$. Să iterăm prin toate perechile de permutări și să le verificăm similaritatea în $O(M)$ este suficient pentru acest subtask.

Subtask 4. Pentru cel de-al patrulea subtask, este nevoie să combinăm cele 2 idei precedente (construcția intelgintă a grafului și folosirea cuplajui) și astfel obținem o complexitate de $O(NM + N^2 + N^2 \log M)$ sau $O(NM + N^2\sqrt{N} + N^2 \log M)$ în funcție de algoritmul de cuplaj folosit.

Subtask 5. Pentru ultimul subtask, putem optimiza construcția grafului. În primul rând, observăm că, știind cel mai lung prefix comun dintre 2 permutări, putem determina în $O(1)$ folosind hashing dacă acestea sunt similare (prima poziție la care diferă determină unic cealaltă poziție știind că acestea diferă prin o transpoziție). Putem încerca să găsim în $O(N^2 + NM)$ cel mai lung prefix comun pentru toate perechile, iar acest lucru este posibil dacă avem permutările în ordine sortată. Având permutările în ordine sortată și notân cu $lcp(i, j)$ cel mai lung prefix comun dintre permutarea i și permutarea j, $(0 < i < j < N)$, observăm că $lcp(i, j) = \min(lcp(i, i+1), lcp(i+1, i), \dots lcp(j-1, j))$, prin urmare ajunge să aplicăm ideea cu căutare binară pentru toate perechile de tip $(i, i+1)$, obținând complexitatea $O(N^2 + N \log M)$. Pentru a obține permutările sortate eficient, putem fie aplica radix sort în $O(NM)$, fie să folosim un algoritm de sortare prin comparare și să aplicăm criteriul de comparare $p[i][lcp(i, j) + 1] < p[j][lcp(i, j) + 1]$. Deoarece $lcp(i, j)$ poate fi

calculat în $O(\log M)$ înainte de precalculare, obținem complexitatea $O(N^2 + N \log N \log M)$ pentru a doua idee de construcție a grafului.

3 Problema Emacs

Propunător: Bogdan Ciobanu

Vom rezolva mai intai cazul cand toate E-urile au aceeasi valoare. Urmatoarele solutii vor adapta ideea, partitionand sirul dupa valorile lui E. Pentru asta, calculam pentru fiecare valoare cat se extinde ca maxim la stanga si la dreapta cu un orice algoritm liniar de "All nearest smaller values". Avand aceste valori calculate, este clar ca pentru o valoare fixa, numarul de subsecvente care au aceast maxim este un produs intre numarul de elemente mai mici la stanga si numarul de elemente mai mici la dreapta. Trebuie luat in calcul ca un maxim nu se va aduna decat o singura data, deci pentru valori egale trebuie ales un reprezentant in numarare, spre exemplu, cea mai la stanga valoare.

Atunci cand $N \leq 5 \cdot 10^4$, exista mai multe solutii simple in stilul min max merge. O observatie utila la problemele de acest tip este ca aceste extinderi ca maxim pot fi vazute ca noduri intr-un arbore cartezian, iar daca iterezi subsecventa mai scurta, adica fie cea din stanga, fie cea din dreapta, atunci complexitatea va fi $O(N \log N)$ (pentru ca sunt de fapt iterezi subarborele cu mai putine noduri), care apoi este inmultit cu complexitatea structurii care tine elementele distincte. In functie de implementare, aceasta solutie poate lua si punctaj maxim, unii participanti spre exemplu au mentinut structurile ca arbori echilibrati care pot fi uniti rapid.

Pentru a extinde solutia cu valori diferite ale lui E, putem sa fixam valoarea lui E si sa iteram pozitiile unde apare. Intre doua pozitii consecutive ne intereseaza sa comprimam sirul, adica sa aflam maximul. Dupa, putem aplica un algoritm similar cu cel mentionat anterior. Exista mai multe moduri de a calcula acest maxim care duc la diferite complexitati. In principiu cel mai familiar mod de a calcula asta este un sparse table sau un arbore de intervale. O alta metoda aplicabila aici care duce la o complexitate mai buna de $O(N\alpha(N))$, fara sa fie prea complicat este cu paduri de multimi disjuncte, in stilul algoritmului lui Tarjan pentru LCA, dar facut implicit: `https://codeforces.com/blog/entry/48994`.

Echipa

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

- Adrian Panaete, profesor Colegiul "A. T. Laurian", Botoșani
- Alexandra Udriștoiu, studentă Universitatea din București
- Alexandru Petrescu, student Universitatea Oxford, Keble College
- Alexa Tudose, studentă Universitatea Oxford, University College
- Andrei Arhire, Universitatea "Alexandru Ioan Cuza" din Iași
- Andrei-Costin Constantinescu, student Universitatea Oxford, Balliol College
- Bogdan Ciobanu, Hudson River Trading.
- Bogdan Ioan Popa, student Universitatea din București.
- Bogdan Iordache, student Universitatea din București
- Bogdan Pop, student Universitatea "Babeș-Bolyai"
- Bogdan Sitaru, student Universitatea Oxford, Hertford College
- Costin-Andrei Oncescu, student Universitatea Oxford, St. John's College
- Emanuel Dicu, student Universitatea Politehnica Bucuresti
- Gabriel Tulbă-Lecu, student Universitatea Politehnica București
- George Chichirim, student Universitatea Oxford, Keble College
- Ionel-Vasile Piț-Rada, profesor Colegiul Național "Traian" Drobeta Turnu Severin
- Lucian Bicsi, doctorand Universitatea din București.
- Marius Nicoli, profesor Colegiul Național "Frații Buzești" Craiova
- Mihai Popescu, student Universitatea "Babeș-Bolyai"
- Stelian Chichirim, student Universitatea din București
- Tamio Vesa Nakajima, student Universitatea Oxford, University College
- Vlad Gavrilă, Google.
- Zoltan Szabo, profesor Liceul Tehnologic "Petru Maior" Reghin / ISJ Mureș Tg. Mureș