DESCRIEREA SOLUTIILOR, OLIMPIADA NATIONALA DE INFORMATICA, BARAJ, ZIUA 2

Problema 1: Secvxor

Propusa de: prof. Mihai Bunget

Subtask 1 (9 puncte). Se generează toate secvenţele *bune* (adică sunt de lungime cel puţin 2 şi elementele de la capetele secvenţei au cel mai mare divizor comun mai mare decât 1), se calculează XOR-ul elementelor fiecărei secvenţe, apoi XOR-ul rezultatelor obţinute pentru aceste secvenţe.

Complexitate: $O(N^3)$.

Subtask 2 (12 puncte). Se generează toate secvenţele, calculând în acelaşi timp şi XOR-ul elementelor secvenţelor bune. Aceste rezultate se adaugă (folosind operaţia XOR) la soluţie.

Complexitate: $O(N^2)$.

Subtask 3 (15 puncte). Deoarece toate elementele sunt pare deducem că toate secvenţele sunt bune. Pentru fiecare element $A[i]$ din şirul dat numărăm din câte secvenţe face parte. Cum $x \oplus x = 0$ deducem că elementul $A[i]$ va fi adăugat la soluţia finală (adică $sol = sol \oplus A_i$) dacă se află într-un număr impar de secvenţe. Numărul secvenţelor în care se află $A[i]$ este $(N-i+1)\cdot(i-1)+N-i$.

Complexitate: $O(N)$.

Subtask 4 (26 puncte). Vom nota cu ps[i] numărul perechilor de elemente egale aflate în şir până la poziţia i şi cu pd[i] numărul celor de la poziţia i la poziţia N. Se parcurge şirul A şi pentru fiecare poziţie i se află numărul secvenţelor bune în care se găseşte elementul A[i]. Acest număr este pd[N]-ps[i-1]-pd[i+1], iar dacă acest număr este impar elementul A[i] va fi adăugat la soluţia finală.

Pentru a calcula ps[i] se adaugă la ps[i-1] frecvenţa elementului A[i] până la indicele i-1. În mod analog se calculează şi pd[i], însă parcurgând şirul de la dreapta la stânga.

Complexitate: $O(N)$.

Subtask 5 (38 puncte). Se foloseşte ciurul lui Eratostene pentru a precalcula factorii primi din descompunerea fiecărui număr mai mic sau egal cu 10^6.

Parcurgem şirul A şi pentru fiecare element $A[i]$ generăm divizorii liberi de pătrate şi determinăm, folosind principiul includerii şi excluderii, numărul perechilor de numere care nu sunt prime cu $A[i]$. Astfel $ps[i]$ va fi suma dintre $ps[i-1]$ şi numărul elementelor situate la stânga lui poziţiei i, care nu sunt prime cu $A[i]$. În mod similar se calculează şi $dp[i]$.

Complexitate: $O(Vmax \cdot \log(Vmax) + 2^8 \cdot N)$.

Problem 2: Trim

Propusa de: stud. Tamio Vesa Nakajima, stud. Bogdan Sitaru, prof. Dan Pracsiu

Notăm cu ϵ șirul vid, și cu ab concatenarea șirurilor lui a și b. Pentru oricare șir a definim $a^0 = \epsilon$ și $a^{n+1} = a^n a$.

Subproblemele 1 și 2. Pentru aceste subprobleme, se generează efectiv întreg șirul (fie programatoric în subproblema 2, fie de mână în subproblema 1), și se afișează bitul cerut.

Subproblema 4. Pentru această subproblema, trebuie să rescriem șirul nostru în altă formă. Observăm că șirul nostru este o concatenare de numere impare în binar, ordonate crescător după numărul de biți de 1, și în caz de egalitate efectiv crescător. De câte ori apare un număr oarecare, să zicem x? Echivalent, pentru câte numere $1 \le y \le 2^N - 1$ avem $\text{trim}(y) = x$? Să presupunem că x are lungimea de l biți. Atunci sunt exact $N - l + 1$ numere y pentru care $\text{trim}(y) = x$, mai exact $y \cdot 2^i$ pentru $0 \le i \le N - l$. În alte cuvinte, șirul începe cu 1 de N ori, apoi 11 de $N-1$ ori, apoi 101 de $N-2$ ori, apoi 1001 de $N-3$ ori, și așa mai departe. Așadar, dacă notăm cu $\binom{n}{k}$ șirurile de n biți dintre care k sunt egali cu 1, șirul nostru este

$$S = 1^N \prod_{b=2}^{N} \prod_{l=b}^{N} \prod_{s \in \binom{l-2}{b-2}} (1s1)^{N-l+1}. \tag{1}$$

În concatenearea aceasta (notată cu produs), mai întâi iterăm prin b (numărul de biți de 1), apoi prin l (lungimea numărului), apoi prin numărul nostru impar $1s1$; numărul acesta apare de $N - l + 1$ ori. Implicit iterăm prin $s \in \binom{l-2}{b-2}$ în ordine crescătoare (sau echivalent lexicografică). Cazul $b = 1$ a fost explicitat separat.

Pentru acest subtask, șirul poate fi generat explicit până la 10^5, și stocat într-un vector, de unde sunt citite efectiv răspunsurile la query-uri.

Soluție completă. Considerăm din nou expresia (1). Definim

$$S_{b,l} = \prod_{s \in \binom{l-2}{b-2}} (1s1)^{N-l+1}.$$

Acest șir are lungimea exact $|S_{b,l}| = \binom{l-2}{b-2}(N - l + 1)l$. Observăm că pentru a calcula cel de-al i-lea bit din $S_{b,l}$, trebuie de fapt să calculăm cel de-al $((i \bmod l) - 1)$-lea bit din cel de-al $\lfloor i/l(N-l+1) \rfloor$ șir din $\binom{l-2}{b-2}$ în ordine lexicografică (sau să afișam 1 imediat dacă $i \bmod l$ este 1 sau l). Acest lucru se poate face, și vom descrie acum cum.

Să zicem, mai general, că vrem să găsim bitul i din al j-lea șir din $\binom{n}{k}$. Mai întâi calculăm primul bit din al j-lea șir din $\binom{n}{k}$. Sunt $\binom{n-1}{k-1}$ șiruri ce încep cu 0; deci dacă $j \le \binom{n-1}{k-1}$, șirul începe cu 0, altfel începe cu 1. Dacă $i = 1$ asta este tot, așadar presupunem că $i > 1$. Dacă șirul începe cu 0, al i-lea bit din al j-lea șir din $\binom{n}{k}$ este același lucru ca al $(i-1)$-lea bit din al j-lea șir din $\binom{n-1}{k}$; altfel dacă începe cu 1 este al $(i-1)$-lea bit din al $(j - \binom{n-1}{k})$-lea șir din $\binom{n-1}{k-1}$. Astfel, aplicând această logică recursiv, putem găsi bitul în timp $O(l)$.

Astfel, pentru orice șir $S_{b,l}$, putem găsi oricare bit din el în $O(l)$; cum $l \le n$, asta e de fapt $O(n)$.

Cum ne ajută asta să rezolvăm problema? Obsevăm, din nou din expresia (1), că

$$S = 1^N \prod_{b=2}^{N} \prod_{l=b}^{N} \prod_{s \in \binom{l-2}{b-2}} (1s1)^{N-l+1} = 1^N \prod_{b=2}^{N} \prod_{l=b}^{N} S_{b,l}.$$

Câti termeni din acest produs se găsesc până la poziția 10^{18}? Observăm că dacă se ajunge la $b = b_0 + 4$, atunci prefixul șirului considerat conține măcar șirul $S_{b_0+4,(b_0+4)/2}$, care are lungimea cel puțin $\binom{b_0}{b_0/2}$. Vrem deci ca $\binom{b_0}{b_0/2} \le 10^{18}$, așadar rezultă că $b_0 \le 32$, de unde rezultă că până pe poziția 10^{18} variabila b nu ajunge decât la 35.

Observăm așadar că șirul S (până la poziția 10^{18}) este un produs de cel mult $N+35N$ termeni, fiecare termen fiind de forma 1 sau $S_{b,l}$. Folosind faptul că $|S_{b,l}| = \binom{l-2}{b-2}(N-l+1)l$, putem găsi efectiv locul unde începe fiecare termen, și să le stocăm în memorie. Pentru a găsi rezultatul, găsim în care termen $S_{b,l}$ ne aflăm cu căutare binară, apoi folosind procedura descrisă anterior găsim bitul cerut din $S_{b,l}$ în $O(N)$.

Această soluție este suficientă pentru a rezolva problema pentru 100 de puncte.

Problema 3: Impiedicat

Propusa de: stud. Costin Oncescu, Adrian Budău

Probleme cere, de fapt, ca pentru un lant $X->Y$ sa se zica de cate ori se schimba maximul; egalitatea se considera si ea o schimbare.

Subtask 1 (11 puncte). Se ia traseul intre nodul X si nodul Y si se simuleaza direct.
Complexitatea este $O(Q \cdot N)$.

Subtask 2 (8 puncte). Fie graful renumerotat ca nodurile sa fie in ordine $1->2->...->N$. Se vor rezolva mai intai toate query-urile cu $X < Y$ si apoi identic cele cu $X > Y$.

Pentru query-uri de tipul $X < Y$ putem calcula (folosind o stiva) pentru fiecare nod i, primul nod la dreapta cu valoarea mai mare sau egala ca cea al nodului i, practic primul nod de care s-ar impiedica Gimi daca ar pleca din i.

Peste aceasta informatie se poate calcula ulterior unde este al 2^{pow} nod de care s-ar impiedica Gimi, ca in stilul algorimtului de calcul al celui de-al K-lea stramos al unui nod in arbore. Cu aceasta informatie se poate apoi cauta binar (tot ca la stramosi) de cate monumente s-ar lovi Gimi pe drumul $X->Y$.

Complexitatea este $O((Q+N)\log N)$.

Subtask 3 (9 puncte). Se rezolva asemanator cu solutia de la lant descrisa anterior, complexitatea ramanand aceeasi. Pentru a afla primul nod cu valoare mai mare sau egala in sus pe arbore se poate tine un arbore de intervale pe stiva din DFS peste care se cauta binar.

Complexitatea ramane aceeasi $O((Q+N)\log N)$.

Subtask 4 (28 de puncte). Vom incerca sa rezolvam lantul cu alta metoda si vom aplica algoritmul pe arbore ulterior.

Sa zicem ca suntem la un nod Y si avem calculat (intr-o structura de date) pentru fiecare nod X din stanga lui Y care este raspunsul pe lantul $X->Y$. Putem raspunde eficinet la toate query-urile care se termina in Y si incep in stanga lui astfel. Fie urmatorul nod dupa Y un nod Z. Se poate observa ca structura calculata pentru Z este foarte asemanatoare cu cea pentru Y. Fie nodul T primul nod din stanga lui Z care are valoare strict mai mare decat cea a nodului Z. Atunci:

- Pentru toate lanturile $X->Z$ cu X din stanga lui T raspunsul e acelasi ca la $X->Y$ deoarece pe drum Gimi sigur se va lovi de nodul T (sau de ceva mai puternic) si dupa va avea suficienta inteligenta sa nu se loveasca de nodul Z.
- Pentru toate lanturile $X->Z$ cu X in dreapta lui T dar in stanga lui Z raspunsul este raspunsul de la $X->Y$ incrementat cu 1 deoarece de oriunde toate nodurile din acest interval au valoare mai mica decat cea a nodului Z.

Existe diferite structuri de date care executa eficient acest increment cu 1 pe interval si query de pozitie: Arbori de Intervale, Arbori indexati binar, etc.

Pentru a rezolva pe arbore trebuie sa calculam aceste noduri T (exact ca la subtaskul 3, poate fi folosita exact aceeasi structura) si sa se rezolve offline toate query-urile din parcurgerea DFS a arborelui. Atunci cand s-a terminat de procesat un nod pe DFS trebuie doar dat undo pe structura la incrementul corespunzator, operatie echivalenta cu adunat -1 pe un interval. Structura aleasa anterior trebuie mentinuta pe adancimea in DFS.

Complexitatea acestei solutii este $O((Q+N)\log N)$.

Subtask 5 (21 de puncte). Se poate face o impartire a drumului dintre $X->Y$ in cel mult $\log N$ lanturi folosind Heavy-light path decomposition. Pe fiecare din aceste lanturi se poate rezolva apoi o problema similara cu cea de la subtask-ul 2 cu diferenta ca Gimi ar o extra valoare de start, eventual mai mare decat cea a primului nod de pe fiecare lant.

Se poate face asta eficient cu o structura care raspunde care este urmatoarea nod cu pe un lant cu valoarea mai mare decat o valoare data X (se poate observa ca este aceeasi structura ca la Subtask-ul 3 pentru a calcula pe arbore in sus informatia dintr-un nod) + solutia de la Subtask-ul 2.

Complexitatea acestei solutii este $O(N \log N + Q \log^2 N)$

Subtask 6 (23 de puncte). Se combina ideile de la solutiile de mai sus pentru a obtine complexitatea cea mai eficienta la aceasta problema.

Fie lantul $X->Y$, el are un varf, cel mai apropiat stramos comun intre X si Y pe drumul spre radacina (radacina poate fi aleasa in orice nod), fie acest nod Z.

- Pe drumul $X->Z$ aplicam exact solutia de la subtask-ul 3.
- Pe drumul $Z->Y$ in jos avem deja o valoare de pornire (de la monumente mai mari de pe drum) asa ca facem ca la subtask-ul 5 sa gasim care e primul nod in jos de la Z la Y unde se va impiedica Gimi, fie acest nod T.
- Drumul $T->Y$ se va rezolva identic ca subtask-ul 4.

Prima parte se poate face pe masura ce se citesc query-urile dar partea a doua si a treia trebuie facute offline, din fericire se pot face deodata in momentul in care DFS parcurgerea este in nodul Y.

Fiecare parte are complexitate $O((Q+N)\log N)$ deci complexitatea finala este $O((Q+N)\log N)$.