Olimpiada Națională de Informatică, Etapa Națională
Proba 2 de baraj
Descrierea Soluțiilor

Comisia Științifică

27 aprilie 2024

Problema 1: Balama

În această problemă, se dă un șir de numere $a_1, \ldots, a_n$, și un număr k. Se consideră toate subsecvențele $a_i, \ldots, a_{i+k-1}$. Acestea se sortează, rezultând șirurile $b_{i1}, \ldots, b_{ik}$. Atunci, se cere, pentru fiecare $j = 1, \ldots, k$, maximul dintre $b_{1j}, \ldots, b_{n-k+1,j}$.

Definim valoarea v_{ix} astfel:

$$v_{xi} = \begin{cases} 1, & a_i \geq x \\ 0 & a_i < x. \end{cases}$$

Definim valoarea s_{xi} astfel

$$s_{xi} = v_{xi} + \cdots + v_{x,i+k-1} = \sum_{j=0}^{k-1} v_{x,i+j}.$$

Fie r_j al j-leq număr din soluție; în alte cuvinte, $r_j = \max_{i=1,\ldots,n-k+1} b_{ij}$.

Observație 1. *$r_j \geq x$ dacă și numai dacă $j \leq \max_i s_{xi}$.*[1]

Demonstrație. Arătăm această echivalență prin a arăta necesitatea și suficiența primei condiții.

Necesitate. Să presupunem că $j \leq \max_i s_{xi}$, adică există i astfel încât $j \leq v_{xi} + \cdots + v_{x,i+k-1}$. În alte cuvinte, subsecvența $a_i, \ldots, a_{i+k-1}$ conține măcar j elemente mai mari sau egale cu x. Prin urmare, subsecvența aceasta arată că $r_j \geq x$.

Suficiență. Să presupunem că $r_j \geq x$. Asta înseamnă că există o subsecvența $a_i, \ldots, a_{i+k-1}$ conține măcar j elemente mai mari sau egale cu x. Așadar, $v_{xi} + \cdots + v_{x,i+k-1} \geq j$, ce implică ca $j \leq \max_i s_{xi}$. □

De aici rezultă imediat următoarea observație.

Observație 2. *r_j este egal cu cel mai mare număr x pentru care $\max_i s_{xi} \leq j$. În simboluri:*

$$r_j = \max\{x \mid \max_i s_{xi} \leq j\}.$$

Mai mult, trebuie considerate doar acele valori x care apar printre valorile $a_1, \ldots, a_n$.

[1] Notă pentru pasionați: aceasta situație, mai exact să existe două funcții f, g unde $f(a) \leq b$ dacă și numai dacă $a \leq g(b)$ se numește o *Conexiune Galois*, și apare în foarte multe contexte diferite în matematică. Noțiuni teoretice mai avansate desigur nu sunt necesare pentru a rezolva această problemă.

Aceste observații ne conduc la următoarea soluție:

1. Parcurgem elementele lui a de la mare la mic. Ținem o structură de date ce menține șirul $t_1, \ldots, t_n$. Să presupunem că suntem la un element $a_i = x$. Scopul nostru va fi ca $t_i = s_{xi}$.
2. Când trecem de la o valoare x la următoarea, observăm că unele elemente din $t_1, \ldots, t_n$ trebuie să crească cu +1. Mai exact, pentru fiecare element $a_i = x$ o subsecvență continuă $t_{\max(i-k+1,0)}, \ldots, t_i$ vor crește cu +1.
3. Când suntem la o valoare x, găsim valoarea t_i de sumă maximă.
4. Ținem o structură de date ce reprezintă șirul $r_1, \ldots, r_k$. Trebuie să creștem valorile de la t_i la k la maximul dintre valoarea precedentă și x; simbolic, $r_j = \max(r_j, x)$ pentru $j \geq t_i$.

Soluția este corectă cum $t_i = s_{xi}$ la momentul x, și conform observației de mai sus. Pentru a o implementa eficient trebuie să folosim structuri de date eficiente. Mai exact, pentru șirul $t_1, \ldots, t_n$ avem nevoie de update-uri ce incrementează o subsecvență, și query-uri ce află maxim-ul întregului șir — ce se poate face cu un arbore de intervale cu propagare lazy în timp logaritmic. Pentru șirul $r_1, \ldots, r_k$, maximizăm sufixe ale șirului, și apoi *după toate update-urile* avem query-uri ce ne cer valorile $r_1, \ldots, r_k$. Pentru a face asta ținem un șir $r'_1, \ldots, r'_k$, la un update la $r_i, \ldots, r_k$ cu x setam $r'_i = \max(r'_i, x)$, și apoi la final calculăm maximele pe prefixe ale lui r'_i pentru a îl găsi pe r_i. Complexitatea finală este $O(n \log n)$.

Problema 2: Piramida

În această problemă se dă o matrice A_{ij} de $N \times M$ caractere, și un șir de caractere $S_0, \ldots, S_{\ell-1}$ (pe care îl indexăm de la 0 pentru convenienţă). Ni se dau mai multe interogări, fiecare interogare constând într-un număr natural k. Pentru a rezolva interogarea, construim șirul $k \times S$, adică S concatenat cu el însuși de k ori. Vrem apoi să găsim numărul de celule din matrice la care s-ar putea ajunge după $k\ell$ pași, începând la oricare celulă, dacă la pasul i valoarea din celulă este egală cu S_i.

Definim $x \oplus y$ ca fiind suma modulo ℓ a lui x și y. Adică, $x \oplus y = (x + y) \bmod \ell$. Să considerăm care este starea noastră în această problemă: Suntem într-o poziție (i, j) la un moment dat, și la un caracter S_t din șirul S. Ele trebuie să satisfacă $S_t = A_{ij}$. De la această stare (i, j, t) putem merge la oricare stare $(i', j', t \oplus 1)$, unde (i, j) și (i', j') sunt vecine, și $S_{t\oplus 1} = A_{i'j'}$.

Observam că stările noastre de mai sus formează un graf G, cu vârfurile (i, j, t) pentru $1 \leq i \leq N, 1 \leq j \leq M$, $0 \leq t < \ell$ unde $A_{ij} = S_t$. Avem o muchie de la (i, j, t) la $(i', j', t \oplus 1)$ dacă și numai dacă (i, j) și (i', j') sunt vecine în matrice. Acum, observăm că se poate ajunge în celula (i, j) după parcurgerea lui $k \times S$ dacă și numai dacă cel lung drum ce ajunge în starea $(i, j, \ell - 1)$ este cel puțin $k\ell$. Asta e pentru că în acest caz, există un lanț de exact $k\ell$ stări ce se termină în starea $(i, j, \ell - 1)$ — ce reprezintă istoricul parcurgerii șirului $k \times S$ și a matricii A care dă în celula (i, j). *Atenție!* Este posibil că cel mai lung drum este ∞, dacă există un ciclu ce poate ajunge la $(i, j, \ell - 1)$. În acest caz se poate ajunge în (i, j) după parcurgerea lui $k \times S$ pentru oricare k.

Așadar, construim graful, și pentru fiecare (i, j) găsim valoarea v_{ij} care este lungimea celui mai lung drum ce intră în starea $(i, j, \ell - 1)$ (sau ∞ dacă există drumuri oricât de lungi). Acest lucru se poate calcula găsind componentele tare conexe ale grafului, apoi sortându-le topologic. Acum, când ne vine o interogare k, este suficient să găsim câte valori v_{ij} sunt mai mare sau egale cu $k\ell$. Asta se poate face sortând valorile v_{ij} și căutând binar.

Pentru a optimiza soluția, putem să nu construim efectiv graful, și să calculăm implicit toate valorile amintite anterior. Soluția finală este de $O(Q \log(NM) + NM\ell)$, unde Q este numărul de interogări.

Problema 3: Echidistant

În această problemă ni se dă un arbore înrădăcinat cu n noduri și valori în noduri (w_u pentru nodul u) și ni se cere să calculăm pentru fiecare subarbore valoarea (suma nodurilor) maximă a unui arbore echidistant care se poate obține din acesta (unde un arbore echidistant este un arbore în care toate frunzele sunt la același nivel).

O abordare asupra problemei este să fixăm nivelul la care vrem să avem frunzele în arborii echidistanți și să calculăm pentru fiecare subarbore valoarea maximă. Vom nota cu $best_u$ valoarea maximă a unui arbore echidistant cu rădăcina în u (adică rezultatul pentru subarborele lui u). Pentru un nivel h fixat vrem să calculăm $best_u$ cu restricția că arborele echidistant de valoare maximă a nodului u are cel puțin o frunză la nivelul h. Așadar, $best_u$ se poate calcula după următoarea formulă:

$$best_u = \begin{cases} w_u + max(best_v) & \text{dacă } max(best_v) < 0 \\ best_u = w_u + \sum_v max(0, best_v) & \text{altfel} \end{cases}$$

, unde v este fiu al nodului u. O soluție implementată naiv va avea complexitate $O(n^2)$.

Putem îmbunătăți soluția precedentă în felul următor: pentru fiecare nivel h, vom construi arborele compresat care conține toate frunzele de pe nivelul h (cunoscut și sub denumirea de arbore virtual). Proprietatea cheie a acestui arbore este că numărul de noduri ale acestui arbore este $O(numarul\ de\ noduri\ la\ nivelul\ h)$. Prin urmare, putem calcula $best_u$ exact ca mai sus.

Totuși, șirul $best$ nu este calculat corect pentru toate nodurile (pot exista noduri care sunt "sărite într-un arbore compresat). Pentru a calcula corect valorile $best_u$ putem să facem o parcurgere în adâncime și să maximizăm valoarea $best_u$ cu $best_v + V_u$, unde v este fiu al nodului u. Această soluție are complexitate $O(n \log n)$.

Pentru a obține o soluție de complexitate $O(n)$, putem optimiza construcția arborilor compresați. Mai exact, fie T arborele compresat care are frunzele la nivelul h. Vom adăuga în T nodurile de la nivelul $h + 1$. Acum, trebuie să ștergem noduri din T în felul următor:

- Dacă subarborele nodului u nu are nicio frunză la nivelul $h + 1$, atunci ștergem subarborele lui u.
- Dacă nodul u (cu tatăl fiind nodul p) are exact un fiu, v, atunci putem compresa nodul u, ștergându-l și adăugând muchia (p, v) în T.
- Dacă nodul u are cel puțin doi fii, atunci nodul se păstrează și nu facem nicio modificare.

După această construcție, vom proceda la fel ca în soluția precedentă.