Shuffle (`shuffle`)

Autor: Alexandru Gheorghieș

Contributor(i): Alexandru Gheorghieș

Soluție

Pentru subtaskul 1 ($N \leq 10, Q \leq 1000$), este suficient să calculăm brut șirul A și să răspundem brut la query-uri. Complexitate: $O(2^N \cdot N + Q \cdot 2^N)$.

Pentru subtaskul 2 ($N \leq 20$), este suficient să calculăm brut șirul A și să răspundem la fiecare query în $O(1)$ folosind sume parțiale. Complexitate: $O(2^N \cdot N + Q)$.

Pentru subtask-urile 3-7, este utilă următoarea observație:

La pasul K ($1 \leq K \leq N$) se vor efectua următoarele modificări pe șirul A:

- $A_0 := A_0 \wedge (2^K - 1)$
- $A_1 := A_1 \wedge (2^K - 1)$

  ...
- $A_{2^N-1} := A_{2^N-1} \wedge (2^K - 1)$

Aici, $\wedge$ reprezintă operația de xor pe biți.

Pentru subtaskul 3 ($R_i - L_i \leq 10$), vom itera prin toate numerele x între L_i și R_i, iar pentru fiecare x vom adăuga $x \wedge (2^1 - 1) \wedge (2^2 - 2) \wedge \ldots \wedge (2^N - 1)$ la răspuns.

Complexitate: $O(Q \cdot (R_i - L_i) \cdot N)$.

Pentru subtaskul 4 ($R_i - L_i \leq 500$), vom precalcula $mask = (2^1 - 1) \wedge (2^2 - 2) \wedge \ldots \wedge (2^N - 1)$.

Vom itera prin toate numerele x între L_i și R_i, iar pentru fiecare x vom adăuga $x \wedge mask$ la răspuns.

Complexitate: $O(Q \cdot (R_i - L_i))$.

Pentru subtask-urile 5-7, vom precalcula $mask = (2^1 - 1) \wedge (2^2 - 2) \wedge \ldots \wedge (2^N - 1)$.

Notăm $l = L_i$ și $r = R_i$.

Va trebui să calculăm într-un mod eficient $\sum_{x=l}^{r}(x \wedge mask)$.

Notăm cu $\text{bit}(x, i)$ valoarea bitului i a numărului x. În $C++$, această funcție poate fi calculată de expresia $(x >> i\&1)$.

De exemplu:

- $\text{bit}(5_{(10)}, 1) = \text{bit}(101_{(2)}, 1) = 0$
- $\text{bit}(12_{(10)}, 3) = \text{bit}(1100_{(2)}, 3) = 1$

Suma de mai sus poate fi rescrisă în felul următor:

$$\sum_{i=1}^{n} 2^i \cdot \Big(\sum_{x=l}^{r} \text{bit(x,i)} \wedge \text{bit(mask,i)} \Big)$$

Notăm cu ftrue(l,r,i) numărul de numere din intervalul $[l, r]$ care au al i-lea bit egal cu 1.

Similar, notăm cu ffalse(l,r,i) numărul de numere din intervalul $[l, r]$ care au al i-lea bit egal cu 0. Observăm că ffalse(l,r,i) $= r - l + 1 -$ ftrue(l,r,i).

Suma de mai sus poate fi rescrisă în felul următor:

$$\sum_{i=1}^{n} 2^i \cdot \begin{cases} \text{ftrue}(l, r, i), & \text{dacă bit(mask,i)} = 0 \\ \text{ffalse}(l, r, i), & \text{dacă bit(mask,i)} = 1 \end{cases}$$

Notăm cu ftrue(r,i) numărul de numere din intervalul $[0, r]$ care au al i-lea bit egal cu 1. Bineînțeles, ftrue(l,r,i) $=$ ftrue$(r, i) -$ ftrue$(l - 1, i)$.

Pentru a calcula ftrue(n, i) în $O(1)$, putem să ne folosim de faptul că funcția bit(n, i) are perioada 2^{i+1}. De exemplu, pentru $i = 2$, funcția bit(n, i) va avea următoarele valori:

$$\underline{0, 0, 0, 0, 1, 1, 1, 1}, \underline{0, 0, 0, 0, 1, 1, 1, 1}, \underline{0, 0, 0, 0, 1, 1, 1, 1}, \underline{0, 0, 0, 0, 1, \ldots}$$

Obținem următoarea formulă pentru ftrue(n, i):

$$\text{ftrue}(n, i) = \begin{cases} 0, & \text{dacă } n < 0 \\ \left\lfloor \frac{n}{2^{i+1}} \right\rfloor \cdot 2^i, & \text{dacă } n \bmod 2^{i+1} < 2^i \\ \left\lfloor \frac{n}{2^{i+1}} \right\rfloor \cdot 2^i + (n \bmod 2^{i+1}) - 2^i + 1, & \text{dacă } n \bmod 2^{i+1} \geq 2^i \end{cases}$$

În funcție de implementare, complexitatea totală va fi $O(Q \cdot N)$ sau $O(Q \cdot N^2)$.

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 $O((T + Q) \log(T + Q))$.