

XORting

Limită de timp: 2 secunde

Limită de memorie: 1024 MiB

Această problemă este una interactivă

Iliyan i-a jucat o farsă lui Jonas: el a ascuns o permutare $p_0, p_1, \dots, p_{N-1}$ a întregilor de la 0 la $N - 1$. Jonas este disperat încercând să o recupereze, și tu ai căzut de acord să-l ajuți.

Iliyan nu va dezvălui permutarea în mod direct, dar a fost de acord să răspundă la întrebări de următorul tip: puteți alege doi indici i și j astfel încât $0 \leq i, j < N$, și el îți va returna XOR pe biți (OR exclusiv) pentru p_i și p_j .

XOR pe biți (OR exclusiv) a două numere, notat cu $\oplus$, realizează operația logică OR exclusiv pe fiecare pereche de biți corespondenți. Rezultatul în fiecare poziție este 1 dacă exact unul dintre biți e 1, dar va fi 0 dacă ambii biți au valoarea 0 sau ambii au valoarea 1. De exemplu, $75 \oplus 29 = 86$, deoarece $1001011_{(2)} \oplus 0011101_{(2)} = 1010110_{(2)}$.

În C++, poți calcula XOR-ul a și b folosind operatorul $\wedge$.

Iliyan este dispus să răspundă la **cel mult** Q întrebări.

Cu toate acestea, întrebările puse nu pot determina întotdeauna permutarea ascunsă în mod unic. Definim o permutare $a_0, a_1, \dots, a_{N-1}$ ca fiind *indistinctibilă* de la p dacă $a_i \oplus a_j = p_i \oplus p_j$ pentru fiecare pereche de indici i și j astfel încât $0 \leq i, j < N$. Fiecare răspuns pe care îl dă Iliyan este identic, indiferent dacă permutarea ascunsă este p sau a , deci nicio secvență de întrebări nu poate face diferența între ele. Reține că p este întotdeauna indistinctibil de el însuși, și că poate să nu existe nicio altă permutare de acest fel.

Sarcina ta este să găsești permutarea cea mai mică lexicografic, indistinctibilă de p .

Iliyan este dispus să răspundă la **cel mult** Q întrebări.

O permutare $x_0, x_1, \dots, x_{N-1}$ este lexicografic mai mică decât $y_0, y_1, \dots, y_{N-1}$ dacă, la prima poziție k în care diferă, $x_k < y_k$. De exemplu, $x = [2, 0, 1, 4, 3]$ este lexicografic mai mică decât $y = [2, 0, 3, 1, 4]$, deoarece prima poziție în care diferă este $k = 2$, și $x_2 = 1 < 3 = y_2$.

Permutarea ascunsă este fixată înainte ca programul tău să înceapă și nu se schimbă ca răspuns la întrebările tale.

Detalii de implementare

Programul tău trebuie să implementeze funcția `solve`:

```
std::vector<int> solve(int N)
```

- N : numărul de elemente ale permutării.

Această funcție este apelată exact T ori, o dată pentru fiecare permutare ascunsă. Trebuie să returnezi permutarea lexicografic cea mai mică care nu se poate distinge de permutarea ascunsă pentru acel apel.

Pentru a comunica cu programul juriului, ți se oferă funcția:

```
int get_xor(int i, int j)
```

De fiecare dată când funcția `get_xor` este apelată, aceasta returnează $p_i \oplus p_j$. Ambii parametri trebuie să fie indici valizi ai permutării. Dacă această condiție nu este îndeplinită, soluția ta va primi verdictul `Output isn't correct: Invalid argument`. Poți presupune că funcția răspunde în timp constant $O(1)$.

Valoarea Q este fixă pentru fiecare subtask și nu este transmisă programului tău.

Restricții

- $1 \leq N_i \leq 2^{20}$ pentru fiecare $1 \leq i \leq T$, unde N_i este valoarea N în al i -lea apel la `solve`
- $1 \leq T \leq 2^{10}$
- $T \cdot \max(N_1, N_2, \dots, N_T) \leq 2^{25}$.
- În timpul al i -lea apel la `solve` într-un test poți pune cel mult Q_i întrebări, unde Q_i este fie N_i fie N_i^2 , în funcție de subtask.

Exemplu

Consideră următoarea interacțiune, în care `solve` este apelat de două ori:

Programul participant	Programul jury
	<code>solve(4)</code>
<code>get_xor(0, 1)</code>	2
<code>get_xor(0, 2)</code>	3
<code>get_xor(0, 3)</code>	1
<code>get_xor(1, 2)</code>	1
<code>get_xor(1, 3)</code>	3
<code>get_xor(2, 3)</code>	2
<code>return {0, 2, 3, 1}</code>	
	<code>solve(1)</code>
<code>return {0}</code>	

Explicația primului apel

Permutarea ascunsă este $[2, 0, 1, 3]$, dar cea mai mică din punct de vedere lexicografic care nu se poate distinge de aceasta este $[0, 2, 3, 1]$. Au fost puse șase întrebări pentru $N_1 = 4$, ceea ce este permis deoarece $Q_1 = N_1^2 = 16$ în acest subtask.

Explicația celui de-al doilea apel

Singura permutare posibilă cu $N_2 = 1$ este $[0]$.

Subtaskuri

Subtask	Puncte	N_i	T	Q_i	Constrângeri suplimentare
0	0	-	-	N_i^2	Exemplul.
1	7	$\leq 2^3$	2^{10}	N_i^2	-
2	18	$\leq 2^7$	2^5	N_i^2	-
3	5	$\leq 2^{11}$	2^5	N_i^2	-
4	5	$\leq 2^{11}$	2^5	N_i	-
5	5	$\leq 2^{18}$	2^5	N_i	$N_i = 2^k$ pentru un anumit întreg k .
6	5	$\leq 2^{18}$	2^5	N_i	N_i este impar.
7	30	$\leq 2^{18}$	2^5	N_i	-
8	25	$\leq 2^{20}$	2^5	N_i	-

Sample grader

Formatul de intrare este următorul:

- linia 1: doi întregi – valorile T și Q_{type} , unde Q_{type} este fie 1 fie 2;
- următoarele $2T$ linii descriu testele:
 - linia $2i$: un întreg – valoarea N_i pentru al i -lea test;
 - linia $2i + 1$: N_i întregi – permutarea $p_0, p_1, \dots, p_{N_i-1}$ pentru al i -lea test.

Dacă $Q_{type} = 1$, atunci $Q_i = N_i$ pentru al i -lea apel la `solve`; dacă $Q_{type} = 2$, atunci $Q_i = N_i^2$.

Formatul de ieșire este următorul:

- linia i : permutarea returnată de programul tău în al i -lea test.