

XORtowanie

Limit czasu: 2 sekundy

Limit pamięci: 1024 MiB

To jest zadanie interaktywne.

Artur zrobił Rafałowi kawę: ukrył permutację $p_0, p_1, \dots, p_{N-1}$ liczb całkowitych od 0 do $N - 1$. Rafał desperacko potrzebuje ją odzyskać, a Ty chcesz mu pomóc.

Artur nie ujawni permutacji, ale zgodził się odpowiadać na następujące pytania: możesz wybrać dwa indeksy i oraz j spełniające $0 \leq i, j < N$, a on poda Ci wynik bitowej operacji XOR (alternatywy wykluczającej) liczb p_i oraz p_j .

Bitowa operacja XOR (alternatywa wykluczająca) dwóch liczb, oznaczana przez $\oplus$, wykonuje operację alternatywy wykluczającej na każdej parze odpowiadających sobie bitów. Wynik na danej pozycji to 1, jeśli dokładnie jeden z bitów to 1, oraz 0, jeśli oba bity to 0 lub oba to 1. Na przykład $75 \oplus 29 = 86$, ponieważ $1001011_{(2)} \oplus 0011101_{(2)} = 1010110_{(2)}$.

W C++ możesz obliczyć XOR dla a i b , używając operatora $\wedge$.

Artur odpowie na **co najwyżej** Q pytań Rafała.

Te pytania jednak nie zawsze pozwalają jednoznacznie określić ukrytą permutację. Definiujemy permutację $a_0, a_1, \dots, a_{N-1}$ jako *nierozróżnialną* od p , jeśli $a_i \oplus a_j = p_i \oplus p_j$ dla każdej pary indeksów i oraz j spełniających $0 \leq i, j < N$. Każda odpowiedź udzielona przez Artura jest identyczna niezależnie od tego, czy ukrytą permutacją jest p , czy a , więc żadna sekwencja pytań Rafała nie pozwala ich odróżnić. Zauważ, że p jest zawsze nierozróżnialna sama od siebie i może być jedyną permutacją nierozróżnialną od p .

Twoim zadaniem jest znalezienie najmniejszej leksykograficznie permutacji nierozróżnialnej od p .

Permutacja $x_0, x_1, \dots, x_{N-1}$ jest mniejsza leksykograficznie od $y_0, y_1, \dots, y_{N-1}$, jeśli na pierwszej pozycji k , na której się różnią, zachodzi $x_k < y_k$. Na przykład $x = [2, 0, 1, 4, 3]$ jest mniejsza leksykograficznie niż $y = [2, 0, 3, 1, 4]$, ponieważ pierwszą pozycją, na której się różnią, jest $k = 2$, oraz $x_2 = 1 < 3 = y_2$.

Ukryta permutacja jest ustalona przed uruchomieniem Twojego programu i nie zmienia się w odpowiedzi na Twoje pytania.

Szczegóły implementacji

Należy zaimplementować funkcję `solve`:

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

- N : liczba elementów permutacji.

Ta funkcja jest wywoływana dokładnie T razy, po jednym razie dla każdej ukrytej permutacji. Musi ona zwrócić najmniejszą leksykograficznie permutację nierozróżnialną od permutacji ukrytej dla tego wywołania.

Aby komunikować się z programem jury, masz do dyspozycji funkcję:

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

Za każdym razem, gdy wywoływana jest funkcja `get_xor`, zwraca ona $p_i \oplus p_j$. Oba argumenty muszą być prawidłowymi indeksami permutacji. Jeśli ten warunek nie zostanie spełniony, Twoje rozwiązanie otrzyma werdykt `Output isn't correct: Invalid argument`. Możesz założyć, że funkcja odpowiada w czasie stałym $O(1)$.

Wartość Q jest ustalona dla każdego podzadania i nie jest przekazywana do Twojego programu.

Ograniczenia

- $1 \leq N_i \leq 2^{20}$ dla każdego $1 \leq i \leq T$, gdzie N_i to wartość N w i -tym wywołaniu funkcji `solve`.
- $1 \leq T \leq 2^{10}$.
- $T \cdot \max(N_1, N_2, \dots, N_T) \leq 2^{25}$.
- Podczas i -tego wywołania funkcji `solve` możesz zadać co najwyżej Q_i pytań, gdzie Q_i to N_i lub N_i^2 , w zależności od podzadania.

Przykład

Rozważmy następującą interakcję, w której funkcja `solve` jest wywoływana dwa razy:

Program zawodnika	Program 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>	

Wyjaśnienie pierwszego wywołania

Ukryta permutacja to $[2, 0, 1, 3]$, ale najmniejszą leksykograficznie permutacją nierozróżnialną od niej jest $[0, 2, 3, 1]$. Zadano sześć pytań dla $N_1 = 4$, co jest dozwolone, ponieważ w tym podzadaniu $Q_1 = N_1^2 = 16$.

Wyjaśnienie drugiego wywołania

Jedyną możliwą permutacją dla $N_2 = 1$ jest $[0]$.

Podzadania

Podzadanie	Punkty	N_i	T	Q_i	Dodatkowe ograniczenia
0	0	-	-	N_i^2	Przykład z zadania.
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	$N_i = 2^k$ dla pewnej liczby całkowitej k .
6	5	$\leq 2^{18}$	2^5	N_i	N_i jest nieparzyste.
7	30	$\leq 2^{18}$	2^5	N_i	-
8	25	$\leq 2^{20}$	2^5	N_i	-

Przykładowy program oceniający

Format wejścia jest następujący:

- pierwszy wiersz: dwie liczby całkowite – wartości T oraz Q_{type} , gdzie Q_{type} wynosi 1 lub 2;
- kolejne $2T$ wierszy opisuje testy:
 - wiersz $2i$: jedna liczba całkowita – wartość N_i dla i -tego testu;
 - wiersz $2i + 1$: N_i liczb całkowitych – permutacja $p_0, p_1, \dots, p_{N_i-1}$ dla i -tego testu.

Jeśli $Q_{type} = 1$, to $Q_i = N_i$ dla i -tego wywołania funkcji `solve`; jeśli $Q_{type} = 2$, to $Q_i = N_i^2$.

Format wyjścia jest następujący:

- wiersz i : permutacja zwrócona przez Twój program w i -tym teście.