

XORtieren

Zeit: 2 Sekunden

Speicherplatz: 1024 MiB

Dies ist eine interaktive Aufgabe.

Iliyan hat Jonas einen Streich gespielt: Er besitzt eine geheime Permutation $p_0, p_1, \dots, p_{N-1}$ von Ganzzahlen zwischen 0 und $N - 1$. Jonas versucht verzweifelt, an diese Permutation zu kommen. Du hast dich bereit erklärt, ihm zu helfen.

Iliyan möchte die Permutation nicht einfach offenlegen. Er hat aber zugestimmt, Fragen der folgenden Form zu beantworten: Du kannst zwei Indizes i und j mit $0 \leq i, j < N$ auswählen, und er verrät dir das bitweise XOR (exklusives Oder) von p_i und p_j .

Das bitweise XOR (exklusives Oder) zweier Zahlen, dargestellt durch $\oplus$, führt die logische exklusive Oder-Operation für alle einander entsprechenden Paare von Bits aus. Das Ergebnis hat eine 1 an jeder Position, an der genau eines der Bits 1 ist. Wenn beide 0 oder 1 sind, hat es an dieser Position eine 0. Beispielsweise gilt $75 \oplus 29 = 86$, da $1001011_{(2)} \oplus 0011101_{(2)} = 1010110_{(2)}$.

In C++ kannst du das XOR von `a` und `b` mit dem Operator `^` berechnen.

Iliyan ist bereit, **maximal** Q Fragen zu beantworten.

Diese Fragen reichen möglicherweise nicht aus, die geheime Permutation eindeutig zu bestimmen. Wir bezeichnen die Permutation $a_0, a_1, \dots, a_{N-1}$ als *ununterscheidbar* von p , wenn $a_i \oplus a_j = p_i \oplus p_j$ für jedes Paar i und j von Indizes mit $0 \leq i, j < N$ gilt. Iliyans Antwort auf jede Frage ist dieselbe, wenn die geheime Permutation p oder a ist. Daher kann keine Abfolge von Fragen die beiden unterscheiden. Beachte, dass p ununterscheidbar von sich selbst ist, und dass möglicherweise keine andere solche Permutation existiert.

Deine Aufgabe ist es, die lexikographisch kleinste, von p ununterscheidbare Permutation zu finden.

Eine Permutation $x_0, x_1, \dots, x_{N-1}$ ist lexikographisch kleiner als $y_0, y_1, \dots, y_{N-1}$, wenn $x_k < y_k$ an der ersten Stelle k gilt, an der sich die beiden unterscheiden. Beispielsweise ist $x = [2, 0, 1, 4, 3]$ lexikographisch kleiner als $y = [2, 0, 3, 1, 4]$, weil sich die beiden Stellen zuerst an Stelle $k = 2$ unterscheiden und $x_2 = 1 < 3 = y_2$ gilt.

Die geheime Permutation wird festgelegt, bevor dein Programm gestartet wird und ändert sich nicht als Reaktion auf deine Fragen.

Implementierungsdetails

Dein Programm muss die Funktion `solve` implementieren:

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

- N : Die Anzahl der Elemente in der Permutation.

Die Funktion wird genau T mal aufgerufen, einmal für jede geheime Permutation. Es muss die lexikographisch kleinste Permutation zurückgeben, die ununterscheidbar von der für diesen Aufruf geheimen Permutation ist.

Um mit dem Programm der Jury zu kommunizieren, wird dir die folgende Funktion bereitgestellt:

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

Jedes Mal, wenn die Funktion `get_xor` aufgerufen wird, gibt sie $p_i \oplus p_j$ zurück. Beide Argumente müssen gültige Indizes der Permutation sein. Wenn diese Bedingung nicht erfüllt wird, erhält deine Lösung das Ergebnis `Output isn't correct: Invalid argument`. Du kannst annehmen, dass diese Funktion in konstanter Zeit $O(1)$ läuft.

Der Wert Q wird für jede Teilaufgabe festgelegt und nicht an dein Programm übergeben.

Beschränkungen

- $1 \leq N_i \leq 2^{20}$ für alle $1 \leq i \leq T$, für die N_i der Wert von N im i -ten Aufruf von `solve` ist
- $1 \leq T \leq 2^{10}$
- $T \cdot \max(N_1, N_2, \dots, N_T) \leq 2^{25}$.
- Während des i -ten Aufrufs von `solve` in einem Test darfst du maximal Q_i Fragen stellen. Dabei ist Q_i in Abhängigkeit von der Teilaufgabe entweder N_i oder N_i^2 .

Beispiel

Betrachte die folgende Interaktion, in der `solve` zweimal aufgerufen wird:

Dein Programm	Programm der 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>	

Erklärung des ersten Aufrufs

Die geheime Permutation ist $[2, 0, 1, 3]$. Die lexikographisch kleinste, davon ununterscheidbare Permutation ist $[0, 2, 3, 1]$. Es wurden sechs Fragen für $N_1 = 4$ gestellt. Das ist erlaubt, weil in dieser Teilaufgabe $Q_1 = N_1^2 = 16$ gilt.

Erklärung des zweiten Aufrufs

Die einzig mögliche Permutation mit $N_2 = 1$ ist $[0]$.

Teilaufgaben

Teilaufgabe	Punkte	N_i	T	Q_i	Zusätzliche Beschränkungen
0	0	-	-	N_i^2	Das Beispiel.
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	$N_i = 2^k$ für eine Ganzzahl k .
6	5	$\leq 2^{18}$	2^5	N_i	N_i ist ungerade.
7	30	$\leq 2^{18}$	2^5	N_i	-
8	25	$\leq 2^{20}$	2^5	N_i	-

Sample grader

Das Eingabeformat ist das folgende:

- Zeile 1: Zwei Ganzzahlen – die Werte von T und Q_{type} , wobei Q_{type} entweder 1 oder 2 ist;
- Die folgenden $2T$ Zeilen beschreiben die Tests:
 - Zeile $2i$: Eine Ganzzahl – der Wert von N_i für den i -ten Test;
 - Zeile $2i + 1$: N_i Ganzzahlen – die Permutation $p_0, p_1, \dots, p_{N_i-1}$ für den i -ten Test;

Ist $Q_{type} = 1$, so gilt $Q_i = N_i$ für den i -ten Aufruf von `solve`; im Falle von $Q_{type} = 2$ gilt $Q_i = N_i^2$.

Das Ausgabeformat ist das folgende:

- Zeile i : Die vom Programm zurückgegebene Permutation für den i -ten Test.