

XORting

Limite de temps : 2 secondes

Limite de mémoire : 1024 Mio

Ce problème est interactif.

Iliyan a joué un tour à Jonas : il a caché une permutation $p_0, p_1, \dots, p_{N-1}$ des entiers de 0 à $N - 1$. Aidez Jonas à la retrouver.

Iliyan ne révélera pas la permutation directement, mais il a accepté de répondre à des questions de la forme : vous choisissez deux indices i et j tels que $0 \leq i, j < N$, et Iliyan donne le XOR bit à bit (OU exclusif) de p_i et p_j .

Le XOR bit à bit (OU exclusif) de deux nombres, noté $\oplus$, effectue l'opération logique OU exclusif sur chaque paire de bits. Le résultat à chaque position est 1 si exactement un des bits est 1, mais sera 0 si les deux sont 0 ou si les deux sont 1. Par exemple, $75 \oplus 29 = 86$, puisque $1001011_{(2)} \oplus 0011101_{(2)} = 1010110_{(2)}$.

En C++, vous pouvez calculer le XOR de `a` et `b` en utilisant l'opérateur `^`.

Iliyan est prêt à répondre à **au plus** Q questions.

Cependant, ces questions ne permettent pas toujours déterminer la permutation cachée de manière unique. Nous définissons une permutation $a_0, a_1, \dots, a_{N-1}$ comme *indistinguishable* de p si $a_i \oplus a_j = p_i \oplus p_j$ pour chaque paire d'indices i et j tels que $0 \leq i, j < N$. Chaque réponse qu'Iliyan donne est identique que la permutation cachée soit p ou a , donc aucune suite de questions ne peut les distinguer. Notez que p est toujours indistinguishable d'elle-même, et qu'il n'y a peut-être aucune autre permutation de ce type.

Votre tâche est de trouver la permutation indistinguishable de p la plus petite dans l'ordre lexicographique.

Une permutation $x_0, x_1, \dots, x_{N-1}$ est plus petite dans l'ordre lexicographique que $y_0, y_1, \dots, y_{N-1}$ si, à la première position k où elles diffèrent, $x_k < y_k$. Par exemple, $x = [2, 0, 1, 4, 3]$ est plus petite dans l'ordre lexicographique que $y = [2, 0, 3, 1, 4]$, car la première position où elles diffèrent est $k = 2$, et $x_2 = 1 < 3 = y_2$.

La permutation cachée est fixée avant le démarrage de votre programme et ne change pas en fonction de vos questions.

Détails d'implémentation

Votre programme doit implémenter la fonction `solve` :

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

- N : le nombre d'éléments de la permutation.

Cette fonction est appelée exactement T fois, une fois pour chaque permutation cachée. Elle doit renvoyer la plus petite permutation, dans l'ordre lexicographique, qui est indistinguishable de la permutation cachée pour cet appel.

Pour communiquer avec le programme du jury, la fonction suivante vous est fournie :

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

À chaque fois que la fonction `get_xor` est appelée, elle renvoie $p_i \oplus p_j$. Les deux arguments doivent être des indices valides de la permutation. Si cette condition n'est pas respectée, votre solution recevra le verdict `Output isn't correct: Invalid argument`. Vous pouvez supposer que la fonction répond en temps constant $O(1)$.

La valeur de Q est fixée pour chaque sous-tâche et n'est pas transmise à votre programme.

Contraintes

- $1 \leq N_i \leq 2^{20}$ pour tout $1 \leq i \leq T$, où N_i est la valeur de N dans le i -ème appel à `solve`.
- $1 \leq T \leq 2^{10}$.
- $T \cdot \max(N_1, N_2, \dots, N_T) \leq 2^{25}$.
- Pendant le i -ème appel à `solve` dans un test, vous pouvez poser au plus Q_i questions, où Q_i est N_i ou N_i^2 , en fonction de la sous-tâche.

Exemple

Considérez l'interaction suivante, dans laquelle `solve` est appelée deux fois :

Programme du participant	Programme du 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>	

Explication du premier appel

La permutation cachée est $[2, 0, 1, 3]$, mais la plus petite permutation indistinguable dans l'ordre lexicographique est $[0, 2, 3, 1]$. Six questions ont été posées pour $N_1 = 4$, ce qui est autorisé car $Q_1 = N_1^2 = 16$ dans cette sous-tâche.

Explication du deuxième appel

La seule permutation possible avec $N_2 = 1$ est $[0]$.

Sous-tâches

Sous-tâche	Points	N_i	T	Q_i	Contraintes supplémentaires
0	0	-	-	N_i^2	Exemple du sujet.
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	$N_i = 2^k$ pour un certain entier k .
6	5	$\leq 2^{18}$	2^5	N_i	N_i est impair.
7	30	$\leq 2^{18}$	2^5	N_i	-
8	25	$\leq 2^{20}$	2^5	N_i	-

Évaluateur d'exemple

Le format d'entrée est le suivant :

- ligne 1 : deux entiers – les valeurs de T et Q_{type} , où Q_{type} vaut soit 1, soit 2 ;
- les $2T$ lignes suivantes décrivent les tests :
 - ligne $2i$: un entier – la valeur de N_i pour le i -ème test ;
 - ligne $2i + 1$: N_i entiers – la permutation $p_0, p_1, \dots, p_{N_i-1}$ pour le i -ème test.

Si $Q_{type} = 1$, alors $Q_i = N_i$ pour le i -ème appel de `solve` ; si $Q_{type} = 2$, alors $Q_i = N_i^2$.

Le format de sortie est le suivant :

- ligne i : la permutation renvoyée par votre programme sur le i -ème test.