Codeforces Round #754 (Div. 2)

A. A.M. Deviation

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

A number a_2 is said to be the arithmetic mean of two numbers a_1 and a_3, if the following condition holds: $a_1 + a_3 = 2 \cdot a_2$.

We define an arithmetic mean deviation of three numbers a_1, a_2 and a_3 as follows: $d(a_1, a_2, a_3) = |a_1 + a_3 - 2 \cdot a_2|$.

Arithmetic means a lot to Jeevan. He has three numbers a_1, a_2 and a_3 and he wants to minimize the arithmetic mean deviation $d(a_1, a_2, a_3)$. To do so, he can perform the following operation any number of times (possibly zero):

- Choose i, j from $\{1, 2, 3\}$ such that $i \neq j$ and increment a_i by 1 and decrement a_j by 1

Help Jeevan find out the minimum value of $d(a_1, a_2, a_3)$ that can be obtained after applying the operation any number of times.

Input

The first line contains a single integer t $(1 \leq t \leq 5000)$ — the number of test cases.

The first and only line of each test case contains three integers a_1, a_2 and a_3 $(1 \leq a_1, a_2, a_3 \leq 10^8)$.

Output

For each test case, output the minimum value of $d(a_1, a_2, a_3)$ that can be obtained after applying the operation any number of times.

Example

input

```
3
3 4 5
2 2 6
1 6 5
```

output

```
0
1
0
```

Note

Note that after applying a few operations, the values of a_1, a_2 and a_3 may become negative.

In the first test case, 4 is already the Arithmetic Mean of 3 and 5.

$$d(3, 4, 5) = |3 + 5 - 2 \cdot 4| = 0$$

In the second test case, we can apply the following operation:

$$(2, 2, 6) \xrightarrow[\text{increment } a_2]{\text{decrement } a_1} (1, 3, 6)$$

$$d(1, 3, 6) = |1 + 6 - 2 \cdot 3| = 1$$

It can be proven that answer can not be improved any further.

In the third test case, we can apply the following operations:

$$(1, 6, 5) \xrightarrow[\text{increment } a_3]{\text{decrement } a_2} (1, 5, 6) \xrightarrow[\text{increment } a_1]{\text{decrement } a_2} (2, 4, 6)$$

$$d(2, 4, 6) = |2 + 6 - 2 \cdot 4| = 0$$

B. Reverse Sort

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ashish has a binary string s of length n that he wants to sort in non-decreasing order.

He can perform the following operation:

1. Choose a subsequence of any length such that its elements are in non-increasing order. Formally, choose any k such that $1 \leq k \leq n$ and any sequence of k indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ such that $s_{i_1} \geq s_{i_2} \geq \ldots \geq s_{i_k}$.
2. Reverse this subsequence in-place. Formally, swap s_{i_1} with s_{i_k}, swap s_{i_2} with $s_{i_{k-1}}$, $\ldots$ and swap $s_{i_{\lfloor k/2 \rfloor}}$ with $s_{i_{\lceil k/2 \rceil + 1}}$ (Here $\lfloor x \rfloor$ denotes the largest integer not exceeding x, and $\lceil x \rceil$ denotes the smallest integer not less than x)

Find the minimum number of operations required to sort the string in non-decreasing order. It can be proven that it is always possible to sort the given binary string in at most n operations.

Input

The first line contains a single integer t $(1 \leq t \leq 1000)$ — the number of test cases. The description of the test cases follows.

The first line of each test case contains an integer n $(1 \leq n \leq 1000)$ — the length of the binary string s.

The second line of each test case contains a binary string s of length n containing only 0s and 1s.

It is guaranteed that the sum of n over all test cases does not exceed 1000.

Output

For each test case output the following:

- The **minimum** number of operations m in the first line ($0 \leq m \leq n$).
- Each of the following m lines should be of the form: k i_1 i_2 ... i_k, where k is the length and $i_1 < i_2 < \ldots < i_k$ are the indices of the chosen subsequence. For them the conditions from the statement must hold.

Example

input

```
3
7
0011111
5
10100
6
001000
```

output

```
0
1
4 1 3 4 5
1
3 3 5 6
```

Note

In the first test case, the binary string is already sorted in non-decreasing order.

In the second test case, we can perform the following operation:

- $k = 4$: choose the indices $\{1, 3, 4, 5\}$
  $\underline{1}\ 0\ \underline{1}\ \underline{0}\ \underline{0} \rightarrow \underline{0}\ 0\ \underline{0}\ \underline{1}\ \underline{1}$

In the third test case, we can perform the following operation:

- $k = 3$: choose the indices $\{3, 5, 6\}$
  $0\ 0\ \underline{1}\ 0\ \underline{0}\ \underline{0} \rightarrow 0\ 0\ \underline{0}\ 0\ \underline{0}\ \underline{1}$

C. Dominant Character

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Ashish has a string s of length n containing only characters 'a', 'b' and 'c'.

He wants to find the length of the smallest substring, which satisfies the following conditions:

- Length of the substring is **at least** 2
- 'a' occurs strictly more times in this substring than 'b'
- 'a' occurs strictly more times in this substring than 'c'

Ashish is busy planning his next Codeforces round. Help him solve the problem.

A string a is a substring of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Input

The first line contains a single integer t $(1 \leq t \leq 10^5)$ — the number of test cases. The description of test cases follows.

The first line of each test case contains a single integer n $(2 \le n \le 10^6)$ — the length of the string s.

The second line of each test case contains a string s consisting only of characters 'a', 'b' and 'c'.

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case, output the length of the smallest substring which satisfies the given conditions or print -1 if there is no such substring.

Example

input

```
3
2
aa
5
cbabb
8
cacabccc
```

output

```
2
-1
3
```

Note

Consider the first test case. In the substring "aa", 'a' occurs twice, while 'b' and 'c' occur zero times. Since 'a' occurs strictly more times than 'b' and 'c', the substring "aa" satisfies the condition and the answer is 2. The substring "a" also satisfies this condition, however its length is not at least 2.

In the second test case, it can be shown that in none of the substrings of "cbabb" does 'a' occur strictly more times than 'b' and 'c' each.

In the third test case, "cacabccc", the length of the smallest substring that satisfies the conditions is 3.

D. Treelabeling

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Eikooc and Sushi play a game.

The game is played on a tree having n nodes numbered 1 to n. Recall that a tree having n nodes is an undirected, connected graph with $n-1$ edges.

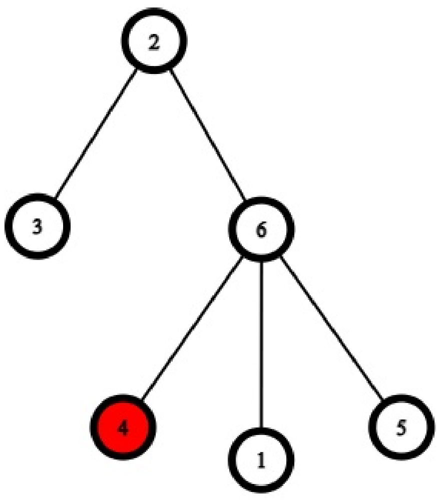
They take turns alternately moving a token on the tree. **Eikooc makes the first move, placing the token** on any node of her choice. Sushi makes the next move, followed by Eikooc, followed by Sushi, and so on. In each turn after the first, a player must move the token to a node u such that

- u is adjacent to the node v the token is currently on
- u has not been visited before
- $u \oplus v \le min(u, v)$

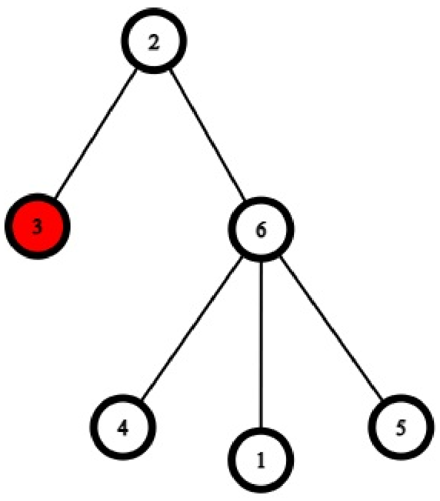
Here $x \oplus y$ denotes the bitwise XOR operation on integers x and y.

Both the players play optimally. The player who is unable to make a move loses.

The following are examples which demonstrate the rules of the game.



Suppose Eikooc starts the game by placing the token at node 4. Sushi then moves the token to node 6, which is unvisited and adjacent to 4. It also holds that $6 \oplus 4 = 2 \le min(6, 4)$. In the next turn, Eikooc moves the token to node 5, which is unvisited and adjacent to 6. It holds that $5 \oplus 6 = 3 \le min(5, 6)$. Sushi has no more moves to play, so she loses.



Suppose Eikooc starts the game by placing the token at node 3. Sushi moves the token to node 2, which is unvisited and adjacent to 3. It also holds that $3 \oplus 2 = 1 \le min(3, 2)$. Eikooc cannot move the token to node 6 since $6 \oplus 2 = 4 \nleq min(6, 2)$. Since Eikooc has no moves to play, she loses.

Before the game begins, Eikooc decides to sneakily relabel the nodes of the tree in her favour. Formally, a relabeling is a permutation p of length n (sequence of n integers wherein each integer from 1 to n occurs exactly once) where p_i denotes the new numbering of node i.

She wants to maximize the number of nodes she can choose in the first turn which will guarantee her a win. Help Eikooc find any relabeling which will help her do so.

Input

The first line contains a single integer t $(1 \le t \le 10^5)$ — the number of test cases. The description of each test case is as follows.

The first line of each test case contains an integer n $(1 \le n \le 2 \cdot 10^5)$ — the number of nodes in the tree.

The next $n - 1$ lines contain two integers u and v $(1 \le u, v \le n; u \ne v)$ — denoting an edge between nodes u and v.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case print any suitable relabeling — a permutation of length n which maximizes the number of nodes that can be chosen in the first turn that guarantee a win for Eikooc. If there are multiple such relabelings, you may print any of them.

Example

input

```
3
1
2
1 2
3
1 2
1 3
```

output

```
1
2 1
1 2 3
```

Note

In the first test case, Eikooc has only one choice. Sushi will have no moves to play after Eikooc chooses this node and Eikooc will win.

In the second test case, $1 \oplus 2 = 3 \nleq min(1, 2)$. Hence, after Eikooc picks either of the nodes, Sushi will have no moves to play and Eikooc will win. Both $\{1, 2\}$ and $\{2, 1\}$ are optimal relabelings.

E. Array Equalizer

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Jeevan has two arrays a and b of size n. He is fond of performing weird operations on arrays. This time, he comes up with two types of operations:

- Choose any i ($1 \le i \le n$) and increment a_j by 1 for every j which is a multiple of i and $1 \le j \le n$.
- Choose any i ($1 \le i \le n$) and decrement a_j by 1 for every j which is a multiple of i and $1 \le j \le n$.

He wants to convert array a into an array b using the minimum total number of operations. However, Jeevan seems to have forgotten the value of b_1. So he makes some guesses. He will ask you q questions corresponding to his q guesses, the i-th of which is of the form:

- If $b_1 = x_i$, what is the minimum number of operations required to convert a to b?

Help him by answering each question.

Input

The first line contains a single integer n $(1 \le n \le 2 \cdot 10^5)$ — the size of arrays a and b.

The second line contains n integers $a_1, a_2, \dots, a_n$ $(1 \le a_i \le 10^6)$.

The third line contains n integers $b_1, b_2, \dots, b_n$ $(1 \le b_i \le 10^6$ for $i \neq 1$; $b_1 = -1$, representing that the value of b_1 is unknown$)$.

The fourth line contains a single integer q $(1 \le q \le 2 \cdot 10^5)$ — the number of questions.

Each of the following q lines contains a single integer x_i $(1 \le x_i \le 10^6)$ — representing the i-th question.

Output

Output q integers — the answers to each of his q questions.

Examples

input

```
2
3 7
-1 5
3
1
4
3
```

output

```
2
4
2
```

input

```
6
2 5 4 1 3 6
-1 4 6 2 3 5
3
1
8
4
```

output

```
10
29
9
```

Note

Consider the first test case.

- $b_1 = 1$: We need to convert $[3, 7] \rightarrow [1, 5]$. We can perform the following operations:

  $$[3,7] \xrightarrow[\text{decrease}]{\text{i = 1}} [2,6] \xrightarrow[\text{decrease}]{\text{i = 1}} [1,5]$$

  Hence the answer is 2.

- $b_1 = 4$: We need to convert $[3, 7] \rightarrow [4, 5]$. We can perform the following operations:

  $$[3,7] \xrightarrow[\text{decrease}]{\text{i = 2}} [3,6] \xrightarrow[\text{decrease}]{\text{i = 2}} [3,5] \xrightarrow[\text{increase}]{\text{i = 1}} [4,6] \xrightarrow[\text{decrease}]{\text{i = 2}} [4,5]$$

  Hence the answer is 4.

- $b_1 = 3$: We need to convert $[3, 7] \rightarrow [3, 5]$. We can perform the following operations:

  $$[3,7] \xrightarrow[\text{decrease}]{\text{i = 2}} [3,6] \xrightarrow[\text{decrease}]{\text{i = 2}} [3,5]$$

  Hence the answer is 2.

F. PalindORme

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

An integer array a of length n is said to be a PalindORme if $(a_1 \mid a_2 \mid \ldots \mid a_i) = (a_{n-i+1} \mid \ldots \mid a_{n-1} \mid a_n)$ **for all** $1 \le i \le n$, where $\mid$ denotes the bitwise OR operation.

An integer array a of length n is considered to be good if its elements can be rearranged to form a PalindORme. Formally, array a is good if there exists a permutation $p_1, p_2, \ldots p_n$ (an array where each integer from 1 to n appears exactly once) for which $a_{p_1}, a_{p_2}, \ldots a_{p_n}$ is a PalindORme.

Find the number of good arrays of length n, consisting only of integers in the range $[0, 2^k - 1]$, and print it modulo some prime m.

Two arrays $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$ are considered to be different if there exists any i $(1 \le i \le n)$ such that $a_i \ne b_i$.

Input

The first and only line of the input contains three integers n, k and m ($1 \le n, k \le 80$, $10^8 < m < 10^9$). It is guaranteed that m is prime.

Output

Print a single integer — the number of good arrays modulo m.

Examples

input
```
1 1 998244353
```
output
```
2
```

input
```
3 2 999999733
```
output
```
40
```

input
```
7 3 796735397
```
output
```
1871528
```

input
```
2 46 606559127
```
output
```
177013
```

Note

In the first sample, both the possible arrays $[0]$ and $[1]$ are good.

In the second sample, some examples of good arrays are:

- $[2, 1, 2]$ because it is already PalindORme.
- $[1, 1, 0]$ because it can rearranged to $[1, 0, 1]$ which is PalindORme

Note that $[1, 1, 0]$, $[1, 0, 1]$ and $[0, 1, 1]$ are all good arrays and are considered to be different according to the definition in the statement.

In the third sample, an example of a good array is $[1, 0, 1, 4, 2, 5, 4]$. It can be rearranged to an array $b = [1, 5, 0, 2, 4, 4, 1]$ which is a PalindORme because:

- $\mathrm{OR}(1, 1) = \mathrm{OR}(7, 7) = 1$
- $\mathrm{OR}(1, 2) = \mathrm{OR}(6, 7) = 5$
- $\mathrm{OR}(1, 3) = \mathrm{OR}(5, 7) = 5$
- $\mathrm{OR}(1, 4) = \mathrm{OR}(4, 7) = 7$
- $\mathrm{OR}(1, 5) = \mathrm{OR}(3, 7) = 7$
- $\mathrm{OR}(1, 6) = \mathrm{OR}(2, 7) = 7$
- $\mathrm{OR}(1, 7) = \mathrm{OR}(1, 7) = 7$

Here $\mathrm{OR}(l, r)$ denotes $b_l \mid b_{l+1} \mid \ldots \mid b_r$