Area Under Path (areaunderpath)

Peter is standing at the origin of the Cartesian plane, and he decided to take a *regular* walk to point (N, M) for some positive integers N and M. In each step of a *regular* walk, Peter must move parallel to one of the axes. During a move, he can go either one unit to the right or one unit up.

Figure 1: Peter on his morning walk on the Cartesian plane.

Formally, a regular walk from $(0, 0)$ to (N, M) is a sequence of points (x_i, y_i) $(0 \leq i \leq N + M)$ such that

- $(x_0, y_0) = (0, 0)$ and $(x_{N+M}, y_{N+M}) = (N, M)$, and
- for each $i = 1, \ldots, N + M$, either $(x_i, y_i) = (x_{i-1} + 1, y_{i-1})$ or $(x_i, y_i) = (x_{i-1}, y_{i-1} + 1)$.

We define the area under a regular walk as the area of the polygon whose vertices in clockwise order are $(0, 0) = (x_0, y_0), (x_1, y_1), \ldots, (x_{N+M}, y_{N+M}) = (N, M)$ and $(N, 0)$.

Given a prime number P and a remainder R, you have to find the number W of *regular* walks from $(0, 0)$ to (N, M) under which the area is congruent to R modulo P. Since the answer can be very large, you have to compute it modulo $10^9 + 7$.

✍ The *modulo* operation (a mod m) can be written in C/C++/Python as `(a % m)` and in Pascal as `(a mod m)`. To avoid the *integer overflow* error, remember to reduce all partial results through the modulus, and not just the final result!
Notice that if $x < 10^9 + 7$*, then* $2x$ *fits into a* C/C++ `int` *and* Pascal `longint`.

☞ Among the attachments of this task you may find a template file `areaunderpath.*` with a sample incomplete implementation.

Input

The input file consists of a single line containing integers N, M, P, R.

Output

The output file must contain a single line consisting of integer W.

Constraints

- $1 \leq N, M \leq 1\,000\,000$.
- $1 \leq P \leq 100$.
- $0 \leq R < P$.
- P is a prime number.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (20 points) $N, M \leq 10$.

– **Subtask 3** (15 points) $N, M \leq 100$.

– **Subtask 4** (30 points) $N \equiv M \equiv 0 \pmod{P}$.

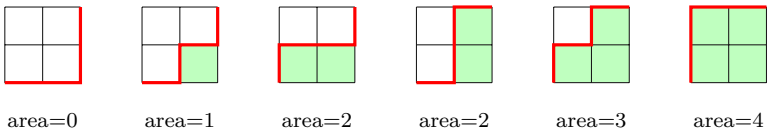
– **Subtask 5** (35 points) No additional limitations.

Examples

input	output
2 2 3 1	2
2 7 5 3	7

Explanation

In the **first sample case**, there are six possible *regular* walks from $(0, 0)$ to (N, M), as shown in the figure below.



The area under the second and the sixth paths are 1 and 4 respectively, both of which give a remainder of 1 when divided by 3.

Precise Average (avg)

Your friend John works at a shop that sells N types of products, numbered from 0 to $N-1$. Product i has a price of P_i bytedollars, where P_i is a positive integer.

The government of Byteland has created a new law for shops. The law states that the average of the prices of all products in a shop should be exactly K, where K is a positive integer. John's boss gave him the task to change the prices of the products so the shop would comply with the new law.

Figure 1: It takes a lot of time to keep the shop tidy! [1]

He has lots of other stuff to do, so he asked you for help: what is the minimum number of products whose prices should be changed? A product's price can be changed to any **positive** integer amount of bytedollars.

☞ Among the attachments of this task you may find a template file `avg.*` with a sample incomplete implementation.

Input

The first line of the input contains two integers N and K. The next line contains N positive integers, $P_0, \ldots, P_{N-1}$.

Output

Output a single integer between 0 and N, inclusive: the answer to the question. It can be proven that it is always possible to change the prices of some products so that the new prices comply with the law.

[1] Photo credits: Diego Delso, delso.photo, License CC-BY-SA

Constraints

- $1 \leq N \leq 200\,000$.
- $1 \leq K \leq 1\,000\,000$.
- $1 \leq P_i \leq 1\,000\,000$ for each $i = 0 \ldots N-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

In this task, you can get partial scores. You will get at least 25% of the points for a subtask if you successfully solve the task when the answer is at most 1. This means that for all test cases in this subtask, you output the correct answer if it's at most 1, otherwise, you may output any integer between 2 and N inclusive.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (25 points) $N \leq 2$.

– **Subtask 3** (35 points) $N \leq 1000$.

– **Subtask 4** (40 points) No additional limitations.

Examples

input	output
2 3 10 6	2
3 9 2 10 1	1

Explanation

In the **first sample case** a possible solution is to change both prices to be 3 bytedollars. It can be proven that changing only one price is not sufficient.

In the **second sample case** a possible solution is to change the first product's price to 16 bytedollars, thus the average will be $\frac{16+10+1}{3} = 9$.

Destroy the Village (barbarian)

A village is being raided by a barbarian, who, despite being on his own, shouldn't be underestimated. The village consists of N houses, numbered from 0 to $N-1$ and arranged in a straight line perpendicular to the shore, with house i being D_i meters away from the shore. The barbarian's strategy is simple: he will raze the house he is in, then move to the closest house that hasn't been razed yet, until all the treasures in the village have been hoarded. If there is more than one house at the minimum distance, the barbarian will move to the one closest to the shore first.

Figure 1: The mighty barbarian.

The villagers have a plan though, which is to hide in the last house that the barbarian will raid to face him when he's most tired. Trying to predict the barbarian's moves is giving them an headache, moreover, the raid may start from any house. In order to help them, you have to find, for each possible starting house i, which house H_i would be destroyed last by the barbarian.

> ☞ Among the attachments of this task you may find a template file `barbarian.*` with a sample incomplete implementation.

Input

The input file consists of:

- a line containing integer N.
- a line containing the N integers $D_0, \ldots, D_{N-1}$.

Output

The output file must contain a single line consisting of the N integers $H_0, \ldots, H_{N-1}$.

Constraints

- $2 \leq N \leq 10^6$.
- $0 \leq D_i \leq 10^{12}$ for each $i = 0 \ldots N-1$.
- $D_i > D_{i-1}$ for each $1 \leq i \leq N-1$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (12 points) $N \leq 100$, $D_i \leq 10^9$, for each $0 \leq i \leq N-1$.

– **Subtask 3** (28 points) $N \leq 1000$.

– **Subtask 4** (35 points) $N \leq 100\,000$.

– **Subtask 5** (25 points) No additional limitations.

Examples

input	output
5 1 4 6 7 10	4 0 4 4 0

Explanation

In the **first sample case**

- When the barbarian starts from house 0, located 1 meter away from the shore, he destroys it, then moves to house 1, at a distance of 3 meters from house 0, then to house 2, then 3. The last house standing is number 4.
- When the barbarian starts from house 1, located 4 meter away from the shore, he destroys it, then moves to house 2, at a distance of 2 meters from house 1, then to house 3, then 4. The last house standing is number 0.
- When the barbarian starts from house 2, located 6 meter away from the shore, he destroys it, then moves to house 3, at a distance of 1 meter from house 2, then to house 1. Note that both house 1 and 4 are at a distance of 3 meters from house 3, but house 1 is closer to the shore. He then moves to house 0. The last house standing is number 4.
- When the barbarian starts from house 3, located 7 meter away from the shore, he destroys it, then moves to house 2, at a distance of 1 meter from house 3, then to house 1, then 0. The last house standing is number 4.
- When the barbarian starts from house 4, located 10 meter away from the shore, he destroys it, then moves to house 3, at a distance of 3 meters from house 4, then to house 2, then 1. The last house standing is number 0.

Binary Chess (`binarychess`)

There is a chess board of R rows and C columns. There are N cells that are occupied by chess pieces, and all other cells are empty. You don't know what exact pieces occupy them, but you know that each piece is either a rook or a bishop. You also know that no rook attacks a bishop, and no bishop attacks a rook.

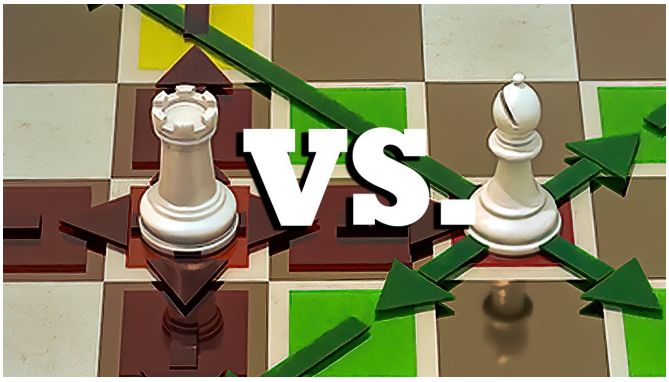


Figure 1: A rook and a bishop, ready to fight.

How many valid arrangements of pieces exist? Since this number might be too big, output it modulo $10^9 + 7$. Two arrangements are considered different if there is at least one cell which is occupied by a different piece.

✍ The *modulo* operation (a mod m) can be written in C/C++/Python as `(a % m)` and in Pascal as `(a mod m)`. To avoid the *integer overflow* error, remember to reduce all partial results through the modulus, and not just the final result!
Notice that if $x < 10^9 + 7$*, then* $2x$ *fits into a* C/C++ `int` *and* Pascal `longint`.

☞ Among the attachments of this task you may find a template file `binarychess.*` with a sample incomplete implementation.

Input

The input file consists of:

- a line containing integers R, C, N.
- N lines, the i-th of which consisting of integers $\texttt{rr}_i$, $\texttt{cc}_i$, which mean that the cell at the r_i-th row and c_i-th column is occupied.

Output

The output file must contain a single line consisting of integer K, the number of piece arrangements modulo $10^9 + 7$, such that no rook attacks a bishop and no bishop attacks a rook.

Constraints

- $1 \leq R, C \leq 10^9$
- $1 \leq N \leq \min(R \cdot C,\ 200\,000)$
- $1 \leq \texttt{rr}_i \leq R$, $1 \leq \texttt{cc}_i \leq C$ for each $i = 0 \ldots N-1$.
- Each cell is occupied by at most one piece (in other words, either $\texttt{rr}_i \neq \texttt{rr}_j$ or $\texttt{cc}_i \neq \texttt{cc}_j$ for $i \neq j$).

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (15 points) $R, C \leq 1000$ and $N \leq \min(R \cdot C,\ 1000)$

– **Subtask 3** (25 points) $R, C \leq 1000$

– **Subtask 4** (30 points) $N \leq \min(R \cdot C,\ 1000)$

– **Subtask 5** (30 points) No additional limitations.

Examples

input	output
4 2 2 1 1 3 2	4
3 3 3 2 1 3 3 1 1	2

Baby Bob's Bracket Sequence (brackets2)

Baby Bob is learning about mathematical expressions. He despises operands and operators, and only likes round brackets.

Figure 1: Baby Bob while learning.

He's got a sequence A of positive integers $A_1, A_2, \ldots, A_N$. He wants to *bracketize* the sequence. A *bracketized* sequence created from A is a sequence of strings $B_1, B_2, \ldots, B_N$ such that each B_i has length A_i, and B_i consists only of either opening brackets "`(`", or closing brackets "`)`", but not both.

For example let $A = (1, 3, 4)$.

- A possible *bracketized* sequence created from A is "`)`", "`)))`", "`((((`".
- The sequence "`)`", "`)()`", "`((((`" is not a *bracketized* sequence created from A, because the second element consists of both opening and closing brackets.
- The sequence "`(`", "`))))`", "`((((`" is not a *bracketized* sequence created from A, because the length of the second element is not 3.
- The sequence "`(`", "`)`" is not a *bracketized* sequence created from A, because it consists of only 2 strings.

Take the string $B_1 + B_2 + \ldots + B_N$ (i.e., concatenate the elements of the *bracketized* sequence). Bob wonders whether he can *bracketize* A so that the resulting string is a valid bracket sequence. A bracket sequence is valid if "`1`" and "`+`" characters can be inserted into it so that it becomes a valid mathematical expression. For example, "`((((()))))`" is a valid bracket sequence if $A = (1, 3, 4)$.

Write a program that finds such a bracket sequence or determines that it's impossible!

> ☞ Among the attachments of this task you may find a template file `brackets.*` with a sample incomplete implementation.

Input

The first line contains the only integer N. The second line contains N integers A_i.

Output

You need to print a valid bracket sequence created from A or -1 if it's not possible to create one.

If there are multiple correct bracket sequences, output any.

Constraints

- $1 \leq N \leq 500$.
- $1 \leq A_i$ for each $i = 0 \ldots N-1$.
- $A_1 + A_2 + \ldots + A_N \leq 50\,000$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (35 points) $N \leq 2$

– **Subtask 3** (35 points) $N \leq 20$ and $A_1 + A_2 + \ldots + A_N \leq 200$.

– **Subtask 4** (30 points) No additional limitations.

Examples

input	output
3 1 3 4	(((())))
4 2 2 1 1	(())()
2 2 1	-1

Explanation

The **first sample case** is explained in the statement.

In the **second sample case** the bracketized sequence is "((", "))", "(", ")".

Excellent Numbers (excellent)

Valerio was recently introduced to the concept of excellent numbers: a positive integer is considered *excellent* if its decimal representation only contains the digits 1 and 5, and it is divisible by 3.
For example, **15** and **111** are *excellent* numbers ($15 = 5 \cdot 3 + 0$ and $111 = 37 \cdot 3 + 0$), while **151** is not ($151 = 50 \cdot 3 + 1$).

Figure 1: 1515 is considered by many an *Angel Number*[2] and also happens to be an *excellent* number!

Valerio is wondering if there exists at least one *excellent* number with exactly N digits. Help him by **finding one**, or by determining that there are no *excellent* numbers with that number of digits!

> ☞ Among the attachments of this task you may find a template file `excellent.*` with a sample incomplete implementation.

Input

The first (and only) line contains the integer N.

Output

You need to write a single line with an integer: an *excellent* number with N digits, if there exists any. If there are multiple solutions, you may print any.

Otherwise, if there is no such number, output `-1`.

Constraints

- $1 \leq N \leq 1\,000\,000$.

[2] An *Angel Number*, in Numerology, is a number with a predictable pattern that is believed to be a sign from the universe.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (33 points) $N \leq 7$.

– **Subtask 3** (33 points) N is even.

– **Subtask 4** (34 points) No additional limitations.

Examples

input	output
2	15

Explanation

In the **first sample case** the number 15 is a valid *excellent* number. 51 is a correct answer, too.

Long Chain (longchain)

You are given a tree with N vertices, numbered from 1 to N (a tree is an undirected connected graph in which there are no cycles). You are asked to partition it into edge-disjoint simple paths such that the length of the shortest one is maximized.

In this task, the length of a path is defined as the number of edges it has.

> ☞ Among the attachments of this task you may find a template file `longchain.*` with a sample incomplete implementation.

Figure 1: A tree with a bunch of vertices.

Input

The first line contains an integer N. The next $N-1$ lines contain 2 integers u_i, v_i, denoting an edge of the tree.

Output

You need to write a single line with an integer: the maximum length of the shortest path over all partitions of the tree into edge-disjoint simple paths.

Constraints

- $2 \leq N \leq 100\,000$.
- $1 \leq u_i, v_i \leq N$.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (20 points) $N \leq 8$.

– **Subtask 3** (25 points) $N \leq 100$.

– **Subtask 4** (30 points) $N \leq 1000$.

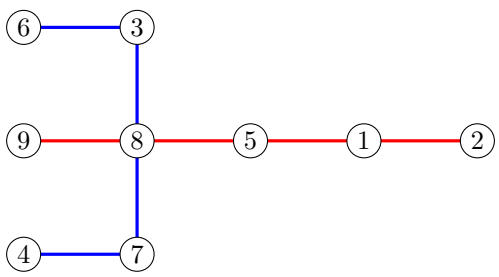
– **Subtask 5** (25 points) No additional limitations.

Examples

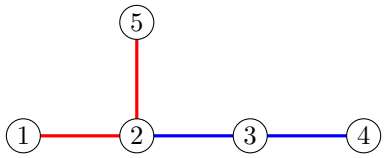
input	output
9 9 8 7 4 7 8 2 1 6 3 5 1 5 8 3 8	4
5 1 2 2 3 3 4 2 5	2
10 7 6 9 6 8 5 4 2 1 3 5 6 1 10 7 10 2 6	4
6 1 2 2 3 3 4 4 5 5 6	5

Explanation

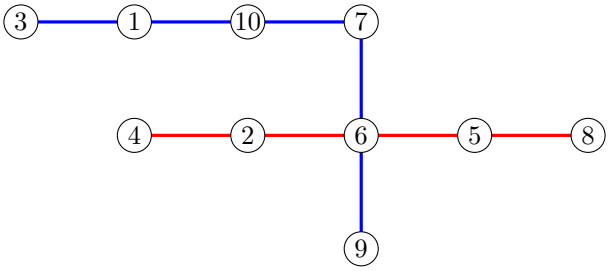
In the **first sample case**, an optimal split would be into 2 edge-disjoint paths, each of length 4: $2-1-5-8-9$ and $4-7-8-3-6$.



In the **second sample case**, an optimal split would be into 2 edge-disjoint paths, each of length 2: $1-2-5$ and $2-3-4$.



In the **third sample case**, an optimal split would be into 2 edge-disjoint paths, one of length 4: $4-2-6-5-8$ and the other one of length 5: $3-1-10-7-6-9$.



In the **fourth sample case**, the whole tree consists of only one path of length 5.

Periodic Words (periodicwords)

A string s is said to be *periodic* if there exists a string t such that s can be obtained by concatenating multiple (at least 2) copies of t. In other words, s is periodic if $s = t + t + \ldots + t$ for some string $t \neq s$, where $+$ is the string concatenation operation.

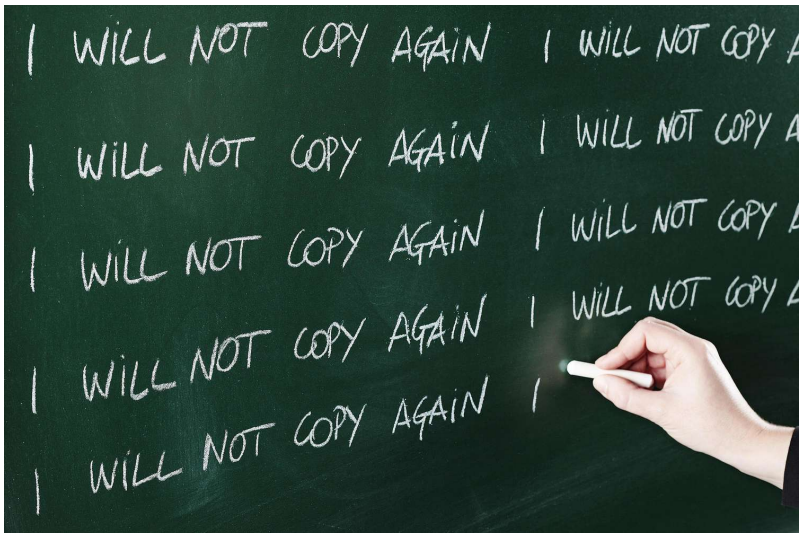


Figure 1: Contestant practising with periodic words.

You are given a string $A = \overline{a_0 a_1 \ldots a_{N-1}}$ of length N and Q queries of the form l_i, r_i. For each query, determine whether the substring $A[l_i \ldots r_i] = \overline{a_{l_i} a_{l_i+1} \ldots a_{r_i}}$ is a periodic string.

☞ Among the attachments of this task you may find a template file `periodicwords.*` with a sample incomplete implementation.

Input

The first line contains an integer N. The second line contains a string S of length N. The third line contains an integer Q. The next Q lines contain the values l_i, r_i, describing the queries.

Output

For each query, print `YES` if the required substring is a periodic string or `NO` if it is not.

Constraints

- $1 \leq N, Q \leq 100\,000$.
- $0 \leq l_i \leq r_i \leq N - 1$.
- The string consist of lowercase letters of the English alphabet.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (20 points) $N, Q \leq 100$.

– **Subtask 3** (25 points) $N, Q \leq 1000$.

– **Subtask 4** (25 points) $N, Q \leq 10\,000$.

– **Subtask 5** (30 points) No additional limitations.

Examples

input	output
14 abacbaabcabccc 5 0 13 0 3 6 11 11 13 6 10	NO NO YES YES NO

Explanation

In the **first query**, it is asked if the whole string is periodic, so the answer is `NO`.

In the **third query**, the substring *abcabc* is periodic, as it can be obtained by concatenating *abc* twice.

Ping-pong (pingpong)

You are attending a table tennis tournament and you decided to note down the results the players achieved. However, a bad misfortune happened and you lost your detailed list of results so you don't know the way the games went anymore.

There is good news though: In order to avoid a complete loss of data, you also noted how many points each player has scored in total during all the sets they played. In addition, you also know that the first player always won in the end.

Figure 1: The games were already played. Can you reconstruct the results?

Now your goal is to reconstruct the way T games turned out, if you know the following details:

- The games have been played using normal table tennis rules, but with a key difference. The set always ends when a player reaches 11 points (even when the score is 11-10, so no tiebreaks exist).
- The match is played using a best of 5 system (the first player to win 3 sets wins the game).
- The first player always won in the end.

Since the games can turn out in many different ways, any such sequence of set scores is acceptable.

☞ Among the attachments of this task you may find a template file `pingpong.*` with a sample incomplete implementation.

Input

The first line of the input will contain T, the number of test cases.

Each test case will contain two integers, A and B, representing the number of points each player obtained during the game.

Output

For each test case, you will either print a way the game could have turned out, with each set written on one line, or `-1 -1` if no such way exists.

Constraints

- $1 \leq T \leq 3000$.
- $1 \leq A, B \leq 60$ for each test case.

Scoring

Your program will be tested against several test cases grouped in subtasks. In order to obtain the score of a subtask, your program needs to correctly solve all of its test cases.

– **Subtask 1** (0 points) Examples.

– **Subtask 2** (30 points) The match can be won by the first player in 3 sets.

– **Subtask 3** (35 points) The match can be won by the first player in 4 sets.

– **Subtask 4** (35 points) No additional limitations.

Examples

input	output
5 33 15 40 29 55 55 39 16 29 54	11 10 11 5 11 0 11 10 11 8 7 11 11 0 -1 -1 11 5 11 0 6 11 11 0 -1 -1

Explanation

In the **first sample case**, the first player can win in 3 sets, with results $11-10$, $11-5$ and $11-0$. Note that in this case, there are multiple ways in which the first player could have won the game.

In the **last sample case**, the total points won by the first player is 29. It can be seen that there is no way in which they could have won 3 sets (and, therefore, the game) with 29 points total.