Codeforces Round #600 (Div. 2)

A. Single Push

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

You're given two arrays $a[1 \dots n]$ and $b[1 \dots n]$, both of the same length n.

In order to perform a *push operation*, you have to choose three integers l, r, k satisfying $1 \le l \le r \le n$ and $k > 0$. Then, you will add k to elements $a_l, a_{l+1}, \dots, a_r$.

For example, if $a = [3, 7, 1, 4, 1, 2]$ and you choose $(l = 3, r = 5, k = 2)$, the array a will become $[3, 7, \underline{3, 6, 3}, 2]$.

You can do this operation **at most once**. Can you make array a equal to array b?

(We consider that $a = b$ if and only if, for every $1 \le i \le n$, $a_i = b_i$)

Input

The first line contains a single integer t ($1 \le t \le 20$) — the number of test cases in the input.

The first line of each test case contains a single integer n ($1 \le n \le 100\,000$) — the number of elements in each array.

The second line of each test case contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 1000$).

The third line of each test case contains n integers $b_1, b_2, \dots, b_n$ ($1 \le b_i \le 1000$).

It is guaranteed that the sum of n over all test cases doesn't exceed 10^5.

Output

For each test case, output one line containing "YES" if it's possible to make arrays a and b equal by performing at most once the described operation or "NO" if it's impossible.

You can print each letter in any case (upper or lower).

Example

input
4 6 3 7 1 4 1 2 3 7 3 6 3 2 5 1 1 1 1 1 1 2 1 3 1 2 42 42 42 42 1 7 6
output
YES NO YES NO

Note

The first test case is described in the statement: we can perform a push operation with parameters $(l = 3, r = 5, k = 2)$ to make a equal to b.

In the second test case, we would need at least two operations to make a equal to b.

In the third test case, arrays a and b are already equal.

In the fourth test case, it's impossible to make a equal to b, because the integer k has to be positive.

B. Silly Mistake

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

output: standard output

The Central Company has an office with a sophisticated security system. There are 10^6 employees, numbered from 1 to 10^6.

The security system logs entrances and departures. The entrance of the i-th employee is denoted by the integer i, while the departure of the i-th employee is denoted by the integer $-i$.

The company has some strict rules about access to its office:

- An employee can enter the office **at most** once per day.
- He obviously can't leave the office if he didn't enter it earlier that day.
- In the beginning and at the end of every day, the office is empty (employees can't stay at night). *It may also be empty at any moment of the day.*

Any array of events satisfying these conditions is called a *valid day*.

Some examples of valid or invalid days:

- $[1, 7, -7, 3, -1, -3]$ is a valid day (1 enters, 7 enters, 7 leaves, 3 enters, 1 leaves, 3 leaves).
- $[2, -2, 3, -3]$ is also a valid day.
- $[2, 5, -5, 5, -5, -2]$ is not a valid day, because 5 entered the office twice during the same day.
- $[-4, 4]$ is not a valid day, because 4 left the office without being in it.
- $[4]$ is not a valid day, because 4 entered the office and didn't leave it before the end of the day.

There are n events $a_1, a_2, \ldots, a_n$, in the order they occurred. This array corresponds to one or more consecutive days. The system administrator erased the dates of events by mistake, but he didn't change the order of the events.

You must partition (to cut) the array a of events into **contiguous subarrays**, which must represent non-empty valid days (or say that it's impossible). Each array element should belong to exactly one contiguous subarray of a partition. Each contiguous subarray of a partition should be a *valid day*.

For example, if $n = 8$ and $a = [1, -1, 1, 2, -1, -2, 3, -3]$ then he can partition it into two contiguous subarrays which are valid days: $a = [1, -1 \mid 1, 2, -1, -2, 3, -3]$.

Help the administrator to partition the given array a in the required way or report that it is impossible to do. Find any required partition, you should not minimize or maximize the number of parts.

Input

The first line contains a single integer n ($1 \le n \le 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-10^6 \le a_i \le 10^6$ and $a_i \neq 0$).

Output

If there is no valid partition, print -1. Otherwise, print any valid partition in the following format:

- On the first line print the number d of days ($1 \le d \le n$).
- On the second line, print d integers $c_1, c_2, \ldots, c_d$ ($1 \le c_i \le n$ and $c_1 + c_2 + \ldots + c_d = n$), where c_i is the number of events in the i-th day.

If there are many valid solutions, you can print **any** of them. You don't have to minimize nor maximize the number of days.

Examples

input
6 1 7 -7 3 -1 -3
output
1 6

input
8 1 -1 1 2 -1 -2 3 -3
output
2 2 6

input
6 2 5 -5 5 -5 -2
output
-1

input

3 -8 1 1
output
-1

Note
In the first example, the whole array is a valid day.

In the second example, one possible valid solution is to split the array into $[1, -1]$ and $[1, 2, -1, -2, 3, -3]$ ($d = 2$ and $c = [2, 6]$). The only other valid solution would be to split the array into $[1, -1]$, $[1, 2, -1, -2]$ and $[3, -3]$ ($d = 3$ and $c = [2, 4, 2]$). Both solutions are accepted.

In the third and fourth examples, we can prove that there exists no valid solution. Please note that the array given in input is **not** guaranteed to represent a coherent set of events.

C. Sweets Eating

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

Tsumugi brought n delicious sweets to the Light Music Club. They are numbered from 1 to n, where the i-th sweet has a sugar concentration described by an integer a_i.

Yui loves sweets, but she can eat at most m sweets each day for health reasons.

Days are 1-indexed (numbered $1, 2, 3, \ldots$). Eating the sweet i at the d-th day will cause a sugar penalty of $(d \cdot a_i)$, as sweets become more sugary with time. A sweet can be eaten at most once.

The total sugar penalty will be the **sum** of the individual penalties of each sweet eaten.

Suppose that Yui chooses exactly k sweets, and eats them in any order she wants. What is the **minimum** total sugar penalty she can get?

Since Yui is an undecided girl, she wants you to answer this question for every value of k between 1 and n.

Input
The first line contains two integers n and m ($1 \le m \le n \le 200\,000$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 200\,000$).

Output
You have to output n integers $x_1, x_2, \ldots, x_n$ on a single line, separed by spaces, where x_k is the minimum total sugar penalty Yui can get if she eats exactly k sweets.

Examples

input
9 2 6 19 3 4 4 2 6 7 8
output
2 5 11 18 30 43 62 83 121

input
1 1 7
output
7

Note
Let's analyze the answer for $k = 5$ in the first example. Here is **one** of the possible ways to eat 5 sweets that minimize total sugar penalty:

- Day 1: sweets 1 and 4
- Day 2: sweets 5 and 3
- Day 3 : sweet 6

Total penalty is $1 \cdot a_1 + 1 \cdot a_4 + 2 \cdot a_5 + 2 \cdot a_3 + 3 \cdot a_6 = 6 + 4 + 8 + 6 + 6 = 30$. We can prove that it's the minimum total sugar penalty Yui can achieve if she eats 5 sweets, hence $x_5 = 30$.

D. Harmonious Graph

time limit per test: 1 second
memory limit per test: 256 megabytes

input: standard input
output: standard output

You're given an undirected graph with n nodes and m edges. Nodes are numbered from 1 to n.

The graph is considered *harmonious* if and only if the following property holds:

- For every triple of integers (l, m, r) such that $1 \le l < m < r \le n$, if there exists a **path** going from node l to node r, then there exists a **path** going from node l to node m.

In other words, in a harmonious graph, if from a node l we can reach a node r through edges ($l < r$), then we should able to reach nodes $(l+1), (l+2), \ldots, (r-1)$ too.

What is the minimum number of edges we need to add to make the graph harmonious?

Input
The first line contains two integers n and m ($3 \le n \le 200\,000$ and $1 \le m \le 200\,000$).

The i-th of the next m lines contains two integers u_i and v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$), that mean that there's an edge between nodes u and v.

It is guaranteed that the given graph is simple (there is no self-loop, and there is at most one edge between every pair of nodes).

Output
Print the minimum number of edges we have to add to the graph to make it harmonious.

Examples

input
14 8 1 2 2 7 3 4 6 3 5 7 3 8 6 8 11 12
output
1

input
200000 3 7 9 9 8 4 5
output
0

Note
In the first example, the given graph is not harmonious (for instance, $1 < 6 < 7$, node 1 can reach node 7 through the path $1 \to 2 \to 7$, but node 1 can't reach node 6). However adding the edge $(2, 4)$ is sufficient to make it harmonious.

In the second example, the given graph is already harmonious.

E. Antenna Coverage

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

The mayor of the Central Town wants to modernize Central Street, represented in this problem by the (Ox) axis.

On this street, there are n antennas, numbered from 1 to n. The i-th antenna lies on the position x_i and has an initial scope of s_i: it covers all integer positions inside the interval $[x_i - s_i; x_i + s_i]$.

It is possible to increment the scope of any antenna by 1, this operation costs 1 coin. We can do this operation as much as we want (multiple times on the same antenna if we want).

To modernize the street, we need to make all integer positions from 1 to m inclusive covered by at least one antenna. Note that it is authorized to cover positions outside $[1; m]$, even if it's not required.

What is the minimum amount of coins needed to achieve this modernization?

Input
The first line contains two integers n and m ($1 \le n \le 80$ and $n \le m \le 100\,000$).

The i-th of the next n lines contains two integers x_i and s_i ($1 \le x_i \le m$ and $0 \le s_i \le m$).

On each position, there is at most one antenna (values x_i are pairwise distinct).

Output
You have to output a single integer: the minimum amount of coins required to make all integer positions from 1 to m inclusive covered by at least one antenna.

Examples

input
3 595 43 2 300 4 554 10
output
281

input
1 1 1 1
output
0

input
2 50 20 0 3 1
output
30

input
5 240 13 0 50 25 60 5 155 70 165 70
output
26

Note
In the first example, here is a possible strategy:

- Increase the scope of the first antenna by 40, so that it becomes $2 + 40 = 42$. This antenna will cover interval $[43 - 42; 43 + 42]$ which is $[1; 85]$
- Increase the scope of the second antenna by 210, so that it becomes $4 + 210 = 214$. This antenna will cover interval $[300 - 214; 300 + 214]$, which is $[86; 514]$
- Increase the scope of the third antenna by 31, so that it becomes $10 + 31 = 41$. This antenna will cover interval $[554 - 41; 554 + 41]$, which is $[513; 595]$

Total cost is $40 + 210 + 31 = 281$. We can prove that it's the minimum cost required to make all positions from 1 to 595 covered by at least one antenna.

Note that positions 513 and 514 are in this solution covered by two different antennas, but it's not important.

—

In the second example, the first antenna already covers an interval $[0; 2]$ so we have nothing to do.

Note that the only position that we needed to cover was position 1; positions 0 and 2 are covered, but it's not important.

F. Cheap Robot

time limit per test: 3 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You're given a simple, undirected, connected, weighted graph with n nodes and m edges.

Nodes are numbered from 1 to n. There are exactly k *centrals* (recharge points), which are nodes $1, 2, \ldots, k$.

We consider a robot moving into this graph, with a battery of capacity c, not fixed by the constructor yet. At any time, the battery

contains an integer amount x of energy between 0 and c inclusive.

Traversing an edge of weight w_i is possible only if $x \geq w_i$, and costs w_i energy points ($x := x - w_i$).

Moreover, when the robot reaches a central, its battery is entirely recharged ($x := c$).

You're given q independent missions, the i-th mission requires to move the robot from central a_i to central b_i.

For each mission, you should tell the minimum capacity required to acheive it.

Input

The first line contains four integers n, m, k and q ($2 \leq k \leq n \leq 10^5$ and $1 \leq m, q \leq 3 \cdot 10^5$).

The i-th of the next m lines contains three integers u_i, v_i and w_i ($1 \leq u_i, v_i \leq n$, $u_i \neq v_i$, $1 \leq w_i \leq 10^9$), that mean that there's an edge between nodes u and v, with a weight w_i.

It is guaranteed that the given graph is simple (there is no self-loop, and there is at most one edge between every pair of nodes) and connected.

The i-th of the next q lines contains two integers a_i and b_i ($1 \leq a_i, b_i \leq k$, $a_i \neq b_i$).

Output

You have to output q lines, where the i-th line contains a single integer : the minimum capacity required to acheive the i-th mission.

Examples

input
10 9 3 1 10 9 11 9 2 37 2 4 4 4 1 8 1 5 2 5 7 3 7 3 2 3 8 4 8 6 13 2 3
output
12

input
9 11 3 2 1 3 99 1 4 5 4 5 3 5 6 3 6 4 11 6 7 21 7 2 6 7 8 4 8 9 3 9 2 57 9 3 2 3 1 2 3
output
38 15

Note

In the first example, the graph is the chain $10 - 9 - 2^C - 4 - 1^C - 5 - 7 - 3^C - 8 - 6$, where centrals are nodes 1, 2 and 3.

For the mission $(2, 3)$, there is only one simple path possible. Here is a simulation of this mission when the capacity is 12.

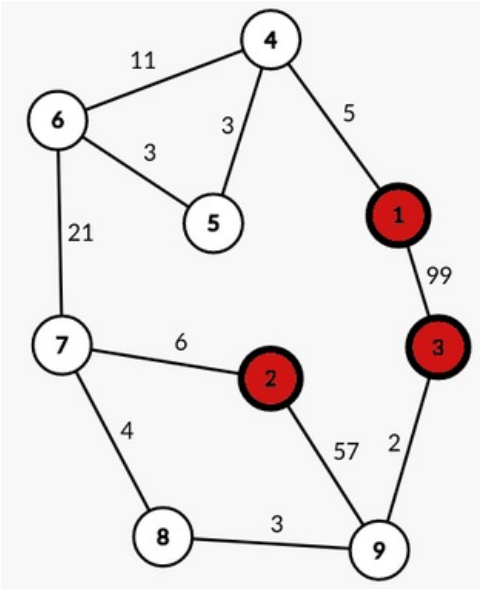
- The robot begins on the node 2, with $c = 12$ energy points.
- The robot uses an edge of weight 4.
- The robot reaches the node 4, with $12 - 4 = 8$ energy points.
- The robot uses an edge of weight 8.
- The robot reaches the node 1 with $8 - 8 = 0$ energy points.
- The robot is on a central, so its battery is recharged. He has now $c = 12$ energy points.
- The robot uses an edge of weight 2.
- The robot is on the node 5, with $12 - 2 = 10$ energy points.
- The robot uses an edge of weight 3.
- The robot is on the node 7, with $10 - 3 = 7$ energy points.
- The robot uses an edge of weight 2.
- The robot is on the node 3, with $7 - 2 = 5$ energy points.
- The robot is on a central, so its battery is recharged. He has now $c = 12$ energy points.

- End of the simulation.

Note that if value of c was lower than 12, we would have less than 8 energy points on node 4, and we would be unable to use the edge $4 \leftrightarrow 1$ of weight 8. Hence 12 is the minimum capacity required to acheive the mission.

—

The graph of the second example is described here (centrals are red nodes):



The robot can acheive the mission $(3, 1)$ with a battery of capacity $c = 38$, using the path
$3 \to 9 \to 8 \to 7 \to 2 \to 7 \to 6 \to 5 \to 4 \to 1$

The robot can acheive the mission $(2, 3)$ with a battery of capacity $c = 15$, using the path $2 \to 7 \to 8 \to 9 \to 3$