CODEFORCES β
Sponsored by Telegram

Codeforces Round #453 (Div. 2)

A. Visiting a Friend

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

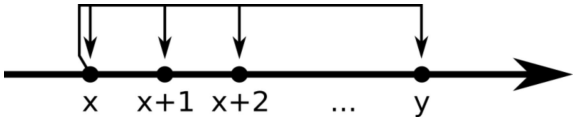
Pig is visiting a friend.

Pig's house is located at point 0, and his friend's house is located at point m on an axis.

Pig can use teleports to move along the axis.

To use a teleport, Pig should come to a certain point (where the teleport is located) and choose where to move: for each teleport there is the rightmost point it can move Pig to, this point is known as the limit of the teleport.

Formally, a teleport located at point x with limit y can move Pig from point x to any point within the segment $[x; y]$, including the bounds.



Determine if Pig can visit the friend using teleports only, or he should use his car.

Input

The first line contains two integers n and m ($1 \le n \le 100$, $1 \le m \le 100$) — the number of teleports and the location of the friend's house.

The next n lines contain information about teleports.

The i-th of these lines contains two integers a_i and b_i ($0 \le a_i \le b_i \le m$), where a_i is the location of the i-th teleport, and b_i is its limit.

It is guaranteed that $a_i \ge a_{i-1}$ for every i ($2 \le i \le n$).

Output

Print "YES" if there is a path from Pig's house to his friend's house that uses only teleports, and "NO" otherwise.

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

Examples

input Copy

```
3 5
0 2
2 4
3 5
```

output

```
YES
```

input Copy

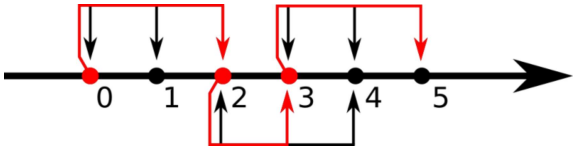
```
3 7
0 4
2 5
6 7
```

output

```
NO
```

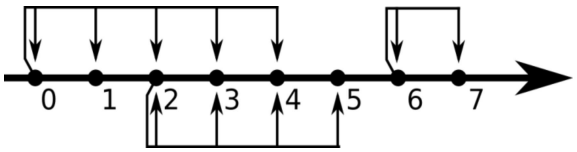
Note

The first example is shown on the picture below:



Pig can use the first teleport from his house (point 0) to reach point 2, then using the second teleport go from point 2 to point 3, then using the third teleport go from point 3 to point 5, where his friend lives.

The second example is shown on the picture below:



You can see that there is no path from Pig's house to his friend's house that uses only teleports.

B. Coloring a Tree

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

You are given a rooted tree with n vertices. The vertices are numbered from 1 to n, the root is the vertex number 1.

Each vertex has a color, let's denote the color of vertex v by c_v. Initially $c_v = 0$.

You have to color the tree into the given colors using the smallest possible number of steps. On each step you can choose a vertex v and a color x, and then color all vectices in the subtree of v (including v itself) in color x. In other words, for every vertex u, such that the path from root to u passes through v, set $c_u = x$.

It is guaranteed that you have to color each vertex in a color different from 0.

You can learn what a rooted tree is using the link: https://en.wikipedia.org/wiki/Tree_(graph_theory).

Input

The first line contains a single integer n ($2 \le n \le 10^4$) — the number of vertices in the tree.

The second line contains $n - 1$ integers $p_2, p_3, \ldots, p_n$ ($1 \le p_i < i$), where p_i means that there is an edge between vertices i and p_i.

The third line contains n integers $c_1, c_2, \ldots, c_n$ ($1 \le c_i \le n$), where c_i is the color you should color the i-th vertex into.

It is guaranteed that the given graph is a tree.

Output

Print a single integer — the minimum number of steps you have to perform to color the tree into given colors.

Examples

input Copy

```
6
1 2 2 1 5
2 1 1 1 1 1
```

output

```
3
```

input Copy

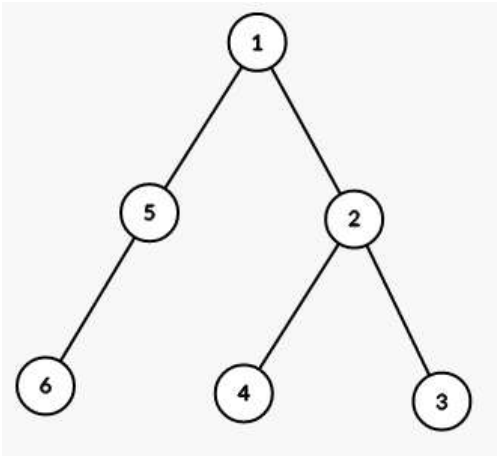
```
7
1 1 2 3 1 4
3 3 1 1 1 2 3
```

output

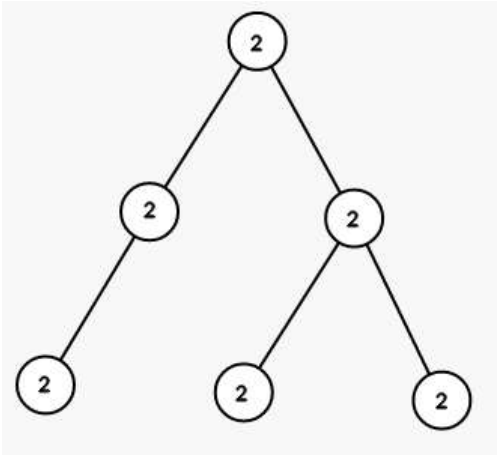
```
5
```

Note

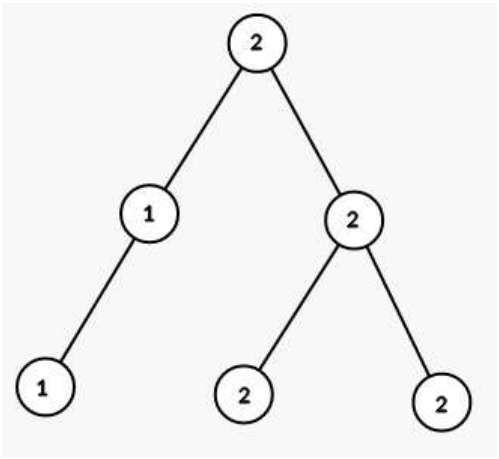
The tree from the first sample is shown on the picture (numbers are vetices' indices):



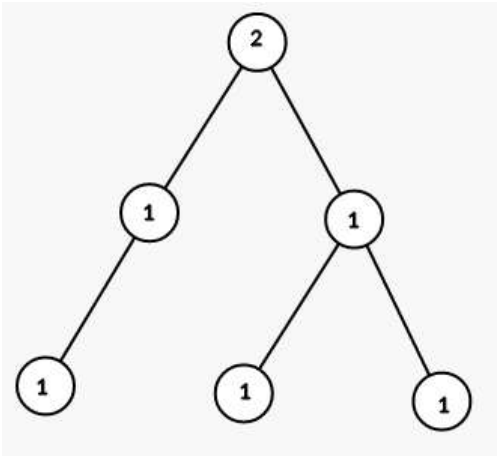
On first step we color all vertices in the subtree of vertex 1 into color 2 (numbers are colors):



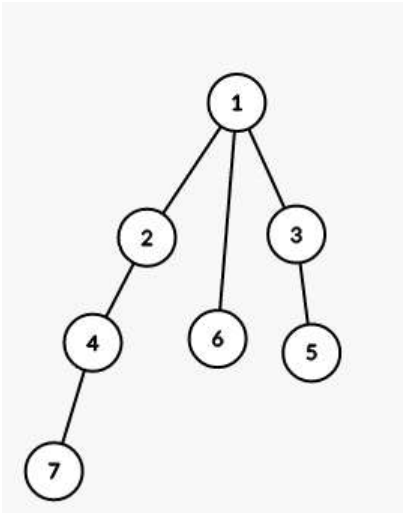
On seond step we color all vertices in the subtree of vertex 5 into color 1:



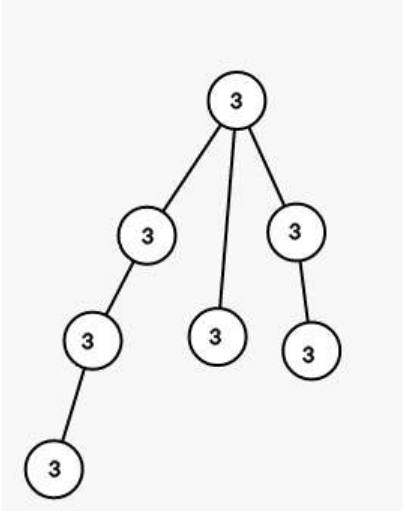
On third step we color all vertices in the subtree of vertex 2 into color 1:



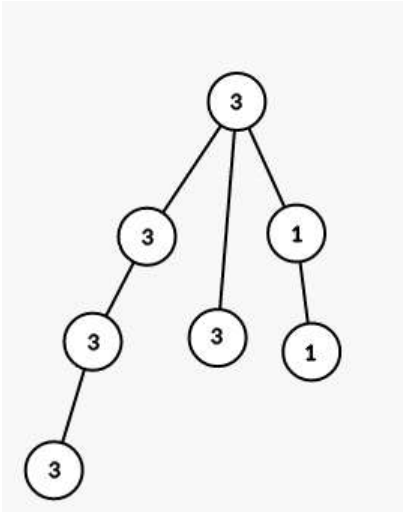
The tree from the second sample is shown on the picture (numbers are vetices' indices):



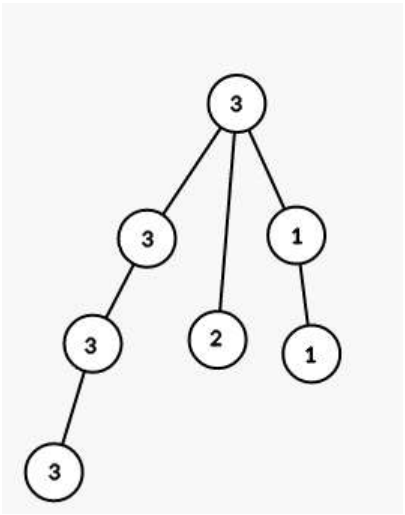
On first step we color all vertices in the subtree of vertex 1 into color 3 (numbers are colors):



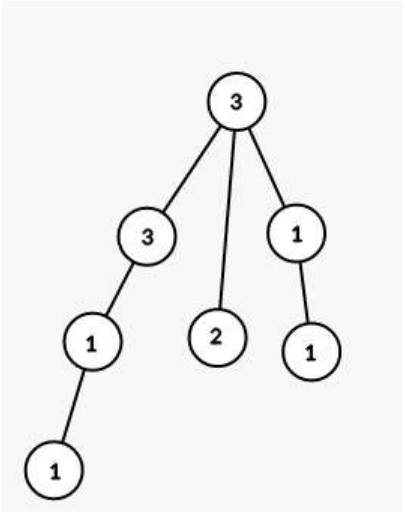
On second step we color all vertices in the subtree of vertex 3 into color 1:



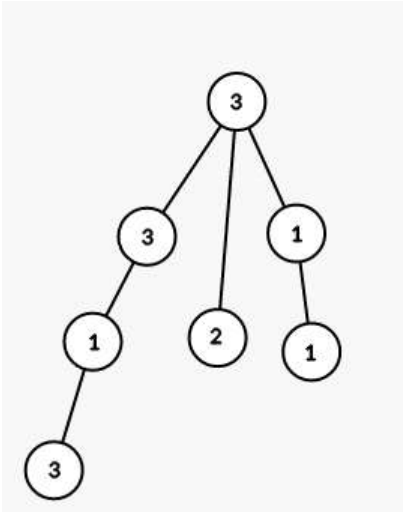
On third step we color all vertices in the subtree of vertex 6 into color 2:



On fourth step we color all vertices in the subtree of vertex 4 into color 1:



On fith step we color all vertices in the subtree of vertex 7 into color 3:



C. Hashing Trees

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

Sasha is taking part in a programming competition. In one of the problems she should check if some rooted trees are isomorphic or not. She has never seen this problem before, but, being an experienced participant, she guessed that she should match trees to some sequences and then compare these sequences instead of trees. Sasha wants to match each tree with a sequence $a_0, a_1, ..., a_h$, where h is the height of the tree, and a_i equals to the number of vertices that are at distance of i edges from root.

Unfortunately, this time Sasha's intuition was wrong, and there could be several trees matching the same sequence. To show it, you need to write a program that, given the sequence a_i, builds two non-isomorphic rooted trees that match that sequence, or determines that there is only one such tree.

Two rooted trees are isomorphic, if you can reenumerate the vertices of the first one in such a way, that the index of the root becomes equal the index of the root of the second tree, and these two trees become equal.

The height of a rooted tree is the maximum number of edges on a path from the root to any other vertex.

Input

The first line contains a single integer h ($2 \le h \le 10^5$) — the height of the tree.

The second line contains $h + 1$ integers — the sequence $a_0, a_1, ..., a_h$ ($1 \le a_i \le 2 \cdot 10^5$). The sum of all a_i does not exceed $2 \cdot 10^5$. It is guaranteed that there is at least one tree matching this sequence.

Output

If there is only one tree matching this sequence, print "perfect".

Otherwise print "ambiguous" in the first line. In the second and in the third line print descriptions of two trees in the following format: in one line print $\sum_{i=0}^{h} a_i$ integers, the k-th of them should be the parent of vertex k or be equal to zero, if the k-th vertex is the root.

These treese should be non-isomorphic and should match the given sequence.

Examples

input Copy

```
2
1 1 1
```

output

```
perfect
```

input Copy

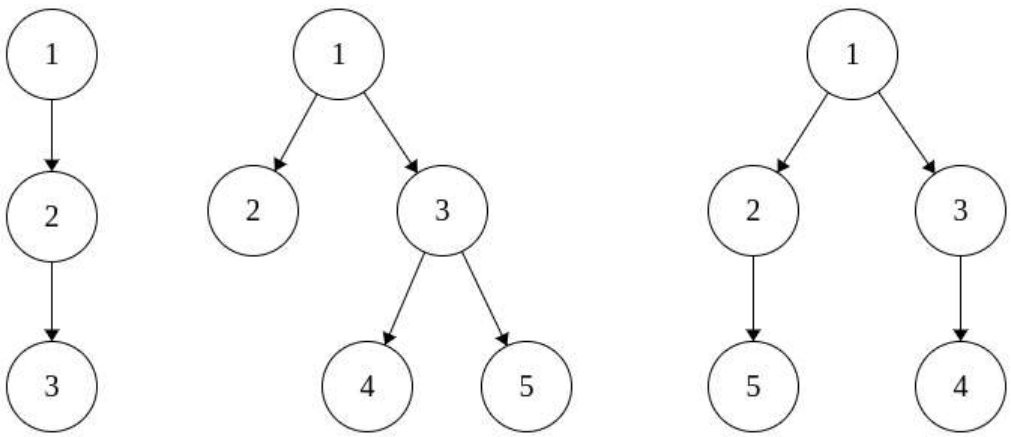
```
2
1 2 2
```

output

```
ambiguous
0 1 1 3 3
0 1 1 3 2
```

Note

The only tree in the first example and the two printed trees from the second example are shown on the picture:



D. GCD of Polynomials

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

Suppose you have two polynomials $A(x) = \sum_{k=0}^{n} a_k x^k$ and $B(x) = \sum_{k=0}^{m} b_k x^k$. Then polynomial $A(x)$ can be uniquely represented in the following way:

$$A(x) = B(x) \cdot D(x) + R(x), \deg R(x) < \deg B(x).$$

This can be done using long division. Here, $\deg P(x)$ denotes the degree of polynomial $P(x)$. $R(x)$ is called the remainder of division of polynomial $A(x)$ by polynomial $B(x)$, it is also denoted as $A \bmod B$.

Since there is a way to divide polynomials with remainder, we can define Euclid's algorithm of finding the greatest common divisor of two polynomials. The algorithm takes two polynomials (A, B). If the polynomial $B(x)$ is zero, the result is $A(x)$, otherwise the result is the value the algorithm returns for pair $(B, A \bmod B)$. On each step the degree of the second argument decreases, so the algorithm works in finite number of steps. But how large that number could be? You are to answer this question.

You are given an integer n. You have to build two polynomials with degrees not greater than n, such that their coefficients are integers not exceeding 1 by their absolute value, the leading coefficients (ones with the greatest power of x) are equal to one, and the described Euclid's algorithm performs exactly n steps finding their greatest common divisor. Moreover, the degree of the first polynomial should be greater than the degree of the second. By a step of the algorithm we mean the transition from pair (A, B) to pair $(B, A \bmod B)$.

Input

You are given a single integer n ($1 \le n \le 150$) — the number of steps of the algorithm you need to reach.

Output

Print two polynomials in the following format.

In the first line print a single integer m ($0 \le m \le n$) — the degree of the polynomial.

In the second line print $m + 1$ integers between -1 and 1 — the coefficients of the polynomial, from constant to leading.

The degree of the first polynomial should be greater than the degree of the second polynomial, the leading coefficients should be equal to 1. Euclid's algorithm should perform exactly n steps when called using these polynomials.

If there is no answer for the given n, print -1.

If there are multiple answer, print any of them.

Examples

input	Copy
1	

output
1 0 1 0 1

input	Copy
2	

output
2 -1 0 1 1 0 1

Note

In the second example you can print polynomials $x^2 - 1$ and x. The sequence of transitions is

$$(x^2 - 1, x) \to (x, -1) \to (-1, 0).$$

There are two steps in it.

E. Bipartite Segments

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

You are given an undirected graph with n vertices. There are no edge-simple cycles with the even length in it. In other words, there are no cycles of even length that pass each edge at most once. Let's enumerate vertices from 1 to n.

You have to answer q queries. Each query is described by a segment of vertices $[l; r]$, and you have to count the number of its subsegments $[x; y]$ ($l \le x \le y \le r$), such that if we delete all vertices except the segment of vertices $[x; y]$ (including x and y) and edges between them, the resulting graph is bipartite.

Input

The first line contains two integers n and m ($1 \le n \le 3 \cdot 10^5$, $1 \le m \le 3 \cdot 10^5$) — the number of vertices and the number of edges in the graph.

The next m lines describe edges in the graph. The i-th of these lines contains two integers a_i and b_i ($1 \le a_i, b_i \le n$; $a_i \ne b_i$), denoting an edge between vertices a_i and b_i. It is guaranteed that this graph does not contain edge-simple cycles of even length.

The next line contains a single integer q ($1 \le q \le 3 \cdot 10^5$) — the number of queries.

The next q lines contain queries. The i-th of these lines contains two integers l_i and r_i ($1 \le l_i \le r_i \le n$) — the query parameters.

Output

Print q numbers, each in new line: the i-th of them should be the number of subsegments $[x; y]$ ($l_i \le x \le y \le r_i$), such that the graph that only includes vertices from segment $[x; y]$ and edges between them is bipartite.

Examples

input Copy

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

output

```
5
5
14
```

input Copy

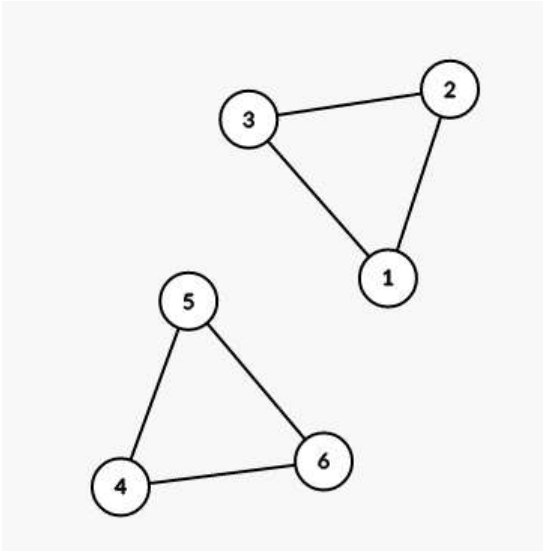
```
8 9
1 2
2 3
3 1
4 5
5 6
6 7
7 8
8 4
7 2
3
1 8
1 4
3 8
```

output

```
27
8
19
```

Note

The first example is shown on the picture below:



For the first query, all subsegments of $[1; 3]$, except this segment itself, are suitable.

For the first query, all subsegments of $[4; 6]$, except this segment itself, are suitable.

For the third query, all subsegments of $[1; 6]$ are suitable, except $[1; 3]$, $[1; 4]$, $[1; 5]$, $[1; 6]$, $[2; 6]$, $[3; 6]$, $[4; 6]$.

The second example is shown on the picture below: