A. Parity

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

You are given an integer n ($n \geq 0$) represented with k digits in base (radix) b. So,

$$n = a_1 \cdot b^{k-1} + a_2 \cdot b^{k-2} + \ldots a_{k-1} \cdot b + a_k.$$

For example, if $b = 17, k = 3$ and $a = [11, 15, 7]$ then $n = 11 \cdot 17^2 + 15 \cdot 17 + 7 = 3179 + 255 + 7 = 3441$.

Determine whether n is even or odd.

Input
The first line contains two integers b and k ($2 \leq b \leq 100$, $1 \leq k \leq 10^5$) — the base of the number and the number of digits.

The second line contains k integers $a_1, a_2, \ldots, a_k$ ($0 \leq a_i < b$) — the digits of n.

The representation of n contains no unnecessary leading zero. That is, a_1 can be equal to 0 only if $k = 1$.

Output
Print "even" if n is even, otherwise print "odd".

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

Examples

input
13 3 3 2 7
output
even

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

input
99 5 32 92 85 74 4
output
odd

input
2 2 1 0
output
even

Note
In the first example, $n = 3 \cdot 13^2 + 2 \cdot 13 + 7 = 540$, which is even.

In the second example, $n = 123456789$ is odd.

In the third example, $n = 32 \cdot 99^4 + 92 \cdot 99^3 + 85 \cdot 99^2 + 74 \cdot 99 + 4 = 3164015155$ is odd.

In the fourth example $n = 2$.

B. Tape

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

You have a long stick, consisting of m segments enumerated from 1 to m. Each segment is 1 centimeter long. Sadly, some segments are broken and need to be repaired.

You have an infinitely long repair tape. You want to cut some pieces from the tape and use them to cover all of the broken segments. To be precise, a piece of tape of integer length t placed at some position s will cover segments $s, s+1, \ldots, s+t-1$.

You are allowed to cover non-broken segments; it is also possible that some pieces of tape will overlap.

Time is money, so you want to cut at most k continuous pieces of tape to cover all the broken segments. What is the minimum total length of these pieces?

Input
The first line contains three integers n, m and k ($1 \leq n \leq 10^5$, $n \leq m \leq 10^9$, $1 \leq k \leq n$) — the number of broken segments, the length of the stick and the maximum number of pieces you can use.

The second line contains n integers $b_1, b_2, \ldots, b_n$ ($1 \leq b_i \leq m$) — the positions of the broken segments. These integers are given in increasing order, that is, $b_1 < b_2 < \ldots < b_n$.

Output
Print the minimum total length of the pieces.

Examples

input
4 100 2 20 30 75 80
output
17

input
5 100 3

1 2 4 60 87

output
6

Note
In the first example, you can use a piece of length 11 to cover the broken segments 20 and 30, and another piece of length 6 to cover 75 and 80, for a total length of 17.

In the second example, you can use a piece of length 4 to cover broken segments 1, 2 and 4, and two pieces of length 1 to cover broken segments 60 and 87.

C. Meaningless Operations

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

Can the greatest common divisor and bitwise operations have anything in common? It is time to answer this question.

Suppose you are given a positive integer a. You want to choose some integer b from 1 to $a - 1$ inclusive in such a way that the greatest common divisor (GCD) of integers $a \oplus b$ and $a \,\&\, b$ is as large as possible. In other words, you'd like to compute the following function:

$$f(a) = \max_{0<b<a} gcd(a \oplus b, a \,\&\, b).$$

Here $\oplus$ denotes the bitwise XOR operation, and $\&$ denotes the bitwise AND operation.

The greatest common divisor of two integers x and y is the largest integer g such that both x and y are divided by g without remainder.

You are given q integers $a_1, a_2, \ldots, a_q$. For each of these integers compute the largest possible value of the greatest common divisor (when b is chosen optimally).

Input
The first line contains an integer q ($1 \le q \le 10^3$) — the number of integers you need to compute the answer for.

After that q integers are given, one per line: $a_1, a_2, \ldots, a_q$ ($2 \le a_i \le 2^{25} - 1$) — the integers you need to compute the answer for.

Output
For each integer, print the answer in the same order as the integers are given in input.

Example

input
3 2 3 5

output
3 1 7

Note
For the first integer the optimal choice is $b = 1$, then $a \oplus b = 3$, $a \,\&\, b = 0$, and the greatest common divisor of 3 and 0 is 3.

For the second integer one optimal choice is $b = 2$, then $a \oplus b = 1$, $a \,\&\, b = 2$, and the greatest common divisor of 1 and 2 is 1.

For the third integer the optimal choice is $b = 2$, then $a \oplus b = 7$, $a \,\&\, b = 0$, and the greatest common divisor of 7 and 0 is 7.

D. Jongmah

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

You are playing a game of Jongmah. You don't need to know the rules to solve this problem. You have n tiles in your hand. Each tile has an integer between 1 and m written on it.

To win the game, you will need to form some number of *triples*. Each triple consists of three tiles, such that the numbers written on the tiles are either all the same or consecutive. For example, $7, 7, 7$ is a valid triple, and so is $12, 13, 14$, but $2, 2, 3$ or $2, 4, 6$ are not. You can only use the tiles in your hand to form triples. Each tile can be used in at most one triple.

To determine how close you are to the win, you want to know the maximum number of triples you can form from the tiles in your hand.

Input
The first line contains two integers integer n and m ($1 \le n, m \le 10^6$) — the number of tiles in your hand and the number of tiles types.

The second line contains integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le m$), where a_i denotes the number written on the i-th tile.

Output
Print one integer: the maximum number of triples you can form.

Examples

input
10 6 2 3 3 3 4 4 4 5 5 6

output
3

input
12 6 1 5 3 3 3 4 3 5 3 2 3 3

output
3

input
13 5 1 1 5 1 2 3 3 2 4 2 3 4 5

output
4

Note
In the first example, we have tiles $2, 3, 3, 3, 4, 4, 4, 5, 5, 6$. We can form three triples in the following way: $2, 3, 4$; $3, 4, 5$; $4, 5, 6$.

Since there are only 10 tiles, there is no way we could form 4 triples, so the answer is 3.

In the second example, we have tiles 1, 2, 3 (7 times), 4, 5 (2 times). We can form 3 triples as follows: $1, 2, 3$; $3, 3, 3$; $3, 4, 5$. One can show that forming 4 triples is not possible.

E. Magic Stones

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

Grigory has n magic stones, conveniently numbered from 1 to n. The *charge* of the i-th stone is equal to c_i.

Sometimes Grigory gets bored and selects some *inner* stone (that is, some stone with index i, where $2 \le i \le n - 1$), and after that *synchronizes* it with neighboring stones. After that, the chosen stone loses its own charge, but acquires the charges from neighboring stones. In other words, its charge c_i changes to $c_i' = c_{i+1} + c_{i-1} - c_i$.

Andrew, Grigory's friend, also has n stones with charges t_i. Grigory is curious, whether there exists a sequence of zero or more *synchronization* operations, which transforms charges of Grigory's stones into charges of corresponding Andrew's stones, that is, changes c_i into t_i for all i?

Input

The first line contains one integer n ($2 \le n \le 10^5$) — the number of magic stones.

The second line contains integers $c_1, c_2, \ldots, c_n$ ($0 \le c_i \le 2 \cdot 10^9$) — the charges of Grigory's stones.

The second line contains integers $t_1, t_2, \ldots, t_n$ ($0 \le t_i \le 2 \cdot 10^9$) — the charges of Andrew's stones.

Output

If there exists a (possibly empty) sequence of *synchronization* operations, which changes all charges to the required ones, print "Yes".

Otherwise, print "No".

Examples

input
4 7 2 4 12 7 15 10 12
output
Yes

input
3 4 4 4 1 2 3
output
No

Note

In the first example, we can perform the following *synchronizations* (1-indexed):

- First, *synchronize* the third stone $[7, 2, \mathbf{4}, 12] \to [7, 2, \mathbf{10}, 12]$.
- Then *synchronize* the second stone: $[7, \mathbf{2}, 10, 12] \to [7, \mathbf{15}, 10, 12]$.

In the second example, any operation with the second stone will not change its charge.

F. Nearest Leaf

time limit per test: 4 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Let's define the Eulerian traversal of a tree (a connected undirected graph without cycles) as follows: consider a depth-first search algorithm which traverses vertices of the tree and enumerates them in the order of visiting (only the first visit of each vertex counts). This function starts from the vertex number 1 and then recursively runs from all vertices which are connected with an edge with the current vertex and are not yet visited in increasing numbers order. Formally, you can describe this function using the following pseudocode:

```
next_id = 1
id = array of length n filled with -1
visited = array of length n filled with false

function dfs(v):
    visited[v] = true
    id[v] = next_id
    next_id += 1
    for to in neighbors of v in increasing order:
        if not visited[to]:
            dfs(to)
```

You are given a weighted tree, the vertices of which were enumerated with integers from 1 to n using the algorithm described above.

A *leaf* is a vertex of the tree which is connected with only one other vertex. In the tree given to you, the vertex 1 is not a leaf. The distance between two vertices in the tree is the sum of weights of the edges on the simple path between them.

You have to answer q queries of the following type: given integers v, l and r, find the shortest distance from vertex v to one of the leaves with indices from l to r inclusive.

Input

The first line contains two integers n and q ($3 \le n \le 500\,000, 1 \le q \le 500\,000$) — the number of vertices in the tree and the number of queries, respectively.

The $(i - 1)$-th of the following $n - 1$ lines contains two integers p_i and w_i ($1 \le p_i < i, 1 \le w_i \le 10^9$), denoting an edge between vertices p_i and i with the weight w_i.

It's guaranteed that the given edges form a tree and the vertices are enumerated in the Eulerian traversal order and that the vertex with index 1 is not a leaf.

The next q lines describe the queries. Each of them contains three integers v_i, l_i, r_i ($1 \le v_i \le n, 1 \le l_i \le r_i \le n$), describing the parameters of the query. It is guaranteed that there is at least one leaf with index x such that $l_i \le x \le r_i$.

Output

Output q integers — the answers for the queries in the order they are given in the input.

Examples

input
5 3 1 10 1 1 3 2 3 3 1 1 5 5 4 5 4 1 2

output
3 0 13

input
5 3 1 1000000000 2 1000000000 1 1000000000 1 1000000000 3 4 5 2 1 5 2 4 5

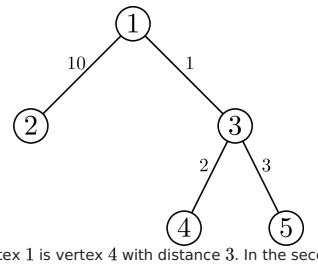
output
3000000000 1000000000 2000000000

input
11 8 1 7 2 1 1 20 1 2 5 6 6 2 6 3 5 1 9 10 9 11 5 1 11 1 1 4 9 4 8 6 1 4 9 7 11 9 10 11 8 1 11 11 4 5

output
8 8 9 16 9 10 0 34

Note

In the first example, the tree looks like this:



In the first query, the nearest leaf for the vertex 1 is vertex 4 with distance 3. In the second query, the nearest leaf for vertex 5 is vertex 5 with distance 0. In the third query the nearest leaf for vertex 4 is vertex 4; however, it is not inside interval $[1, 2]$ of the query. The only leaf in interval $[1, 2]$ is vertex 2 with distance 13 from vertex 4.

G. Tree-Tac-Toe

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

The tic-tac-toe game is starting on a tree of n vertices. Some vertices are already colored in white while the remaining are uncolored.

There are two players — white and black. The players make moves alternatively. The white player starts the game. In his turn, a player must select one uncolored vertex and paint it in his color.

The player wins if he paints some path of three vertices in his color. In case all vertices are colored and neither player won, the game ends in a draw.

Could you please find who will win the game or whether it ends as a draw, assuming both players play optimally?

Input

The first line contains a single integer T ($1 \le T \le 50\,000$) — the number of test cases. Then descriptions of T test cases follow.

The first line of each test contains a single integer n ($1 \le n \le 5 \cdot 10^5$) — the number of vertices in the tree.

Each of the following $n - 1$ lines contains integers v, u ($1 \le v, u \le n$) denoting an edge of the tree connecting vertices v and u.

The last line of a test case contains a string of letters 'W' (for white) and 'N' (for not colored) of length n denoting already colored vertices. Vertexes already colored in white are denoted as 'W'.

It's guaranteed that the given edges form a tree, that there is at least one uncolored vertex and that there is no path of three white vertices.

It's guaranteed that sum of all n among all test cases is at most $5 \cdot 10^5$.

Output

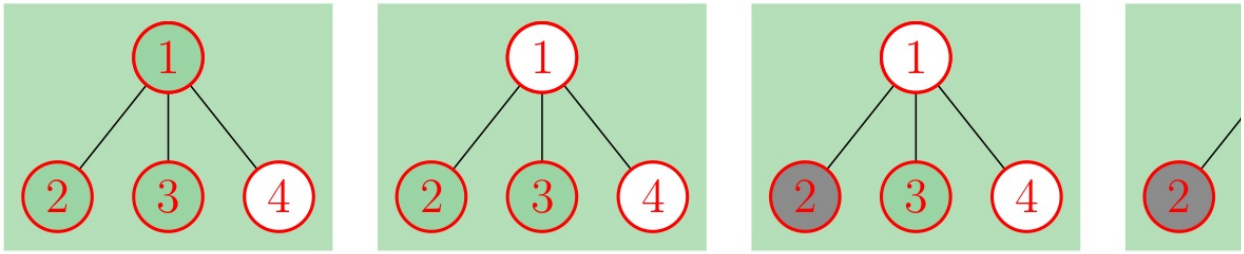
For every test case, print either "White", "Draw" or "Black", depending on the result of the game.

Example

input

2 4 1 2 1 3 1 4 NNNW 5 1 2 2 3 3 4 4 5 NNNNN
output
White Draw

Note
In the first example, vertex 4 is already colored in white. The white player can win by coloring the vertex 1 in white first and the remaining vertex on his second turn. The process is illustrated with the pictures below.



In the second example, we can show that no player can enforce their victory.

H. Modest Substrings

time limit per test: 5 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

You are given two integers l and r.

Let's call an integer x *modest*, if $l \le x \le r$.

Find a string of length n, consisting of digits, which has the largest possible number of substrings, which make a modest integer. Substring having leading zeros are not counted. If there are many answers, find lexicographically smallest one.

If some number occurs multiple times as a substring, then in the counting of the number of modest substrings it is counted multiple times as well.

Input
The first line contains one integer l ($1 \le l \le 10^{800}$).

The second line contains one integer r ($l \le r \le 10^{800}$).

The third line contains one integer n ($1 \le n \le 2\,000$).

Output
In the first line, print the maximum possible number of modest substrings.

In the second line, print a string of length n having exactly that number of modest substrings.

If there are multiple such strings, print the lexicographically smallest of them.

Examples

input
1 10 3
output
3 101

input
1 11 3
output
5 111

input
12345 12346 6
output
1 012345

Note
In the first example, string «101» has modest substrings «1», «10», «1».

In the second example, string «111» has modest substrings «1» (3 times) and «11» (2 times).