Codeforces Round #691 (Div. 1)

A. Row GCD

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given two positive integer sequences $a_1, \ldots, a_n$ and $b_1, \ldots, b_m$. For each $j = 1, \ldots, m$ find the greatest common divisor of $a_1 + b_j, \ldots, a_n + b_j$.

Input

The first line contains two integers n and m ($1 \le n, m \le 2 \cdot 10^5$).

The second line contains n integers $a_1, \ldots, a_n$ ($1 \le a_i \le 10^{18}$).

The third line contains m integers $b_1, \ldots, b_m$ ($1 \le b_j \le 10^{18}$).

Output

Print m integers. The j-th of them should be equal to GCD$(a_1 + b_j, \ldots, a_n + b_j)$.

Example

input
4 4 1 25 121 169 1 2 7 23
output
2 3 8 24

B. Glass Half Spilled

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

There are n glasses on the table numbered $1, \ldots, n$. The glass i can hold up to a_i units of water, and currently contains b_i units of water.

You would like to choose k glasses and collect as much water in them as possible. To that effect you can pour water from one glass to another as many times as you like. However, because of the glasses' awkward shape (and totally unrelated to your natural clumsiness), each time you try to transfer any amount of water, half of the amount is spilled on the floor.

Formally, suppose a glass i currently contains c_i units of water, and a glass j contains c_j units of water. Suppose you try to transfer x units from glass i to glass j (naturally, x can not exceed c_i). Then, $x/2$ units is spilled on the floor. After the transfer is done, the glass i will contain $c_i - x$ units, and the glass j will contain $\min(a_j, c_j + x/2)$ units (excess water that doesn't fit in the glass is also spilled).

Each time you transfer water, you can arbitrarlly choose from which glass i to which glass j to pour, and also the amount x transferred can be any positive real number.

For each $k = 1, \ldots, n$, determine the largest possible total amount of water that can be collected in arbitrarily chosen k glasses after transferring water between glasses zero or more times.

Input

The first line contains a single integer n ($1 \le n \le 100$) — the number of glasses.

The following n lines describe the glasses. The i-th of these lines contains two integers a_i and b_i ($0 \le b_i \le a_i \le 100$, $a_i > 0$) — capacity, and water amount currently contained for the glass i, respectively.

Output

Print n real numbers — the largest amount of water that can be collected in $1, \ldots, n$ glasses respectively. Your answer will be accepted if each number is within 10^{-9} absolute or relative tolerance of the precise answer.

Example

input
3

6 5
6 5
10 2

output

7.0000000000 11.0000000000 12.0000000000

Note

In the sample case, you can act as follows:

- for $k = 1$, transfer water from the first two glasses to the third one, spilling $(5 + 5)/2 = 5$ units and securing $2 + (5 + 5)/2 = 7$ units;
- for $k = 2$, transfer water from the third glass to any of the first two, spilling $2/2 = 1$ unit and securing $5 + 5 + 2/2 = 11$ units;
- for $k = 3$, do nothing. All $5 + 5 + 2 = 12$ units are secured.

C. Latin Square

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a square matrix of size n. Every row and every column of this matrix is a permutation of $1, 2, \ldots, n$. Let $a_{i,j}$ be the element at the intersection of i-th row and j-th column for every $1 \leq i, j \leq n$. Rows are numbered $1, \ldots, n$ top to bottom, and columns are numbered $1, \ldots, n$ left to right.

There are six types of operations:

- R: cyclically shift all columns to the right, formally, set the value of each $a_{i,j}$ to $a_{i,((j-2) \bmod n)+1}$;
- L: cyclically shift all columns to the left, formally, set the value of each $a_{i,j}$ to $a_{i,(j \bmod n)+1}$;
- D: cyclically shift all rows down, formally, set the value of each $a_{i,j}$ to $a_{((i-2) \bmod n)+1,j}$;
- U: cyclically shift all rows up, formally, set the value of each $a_{i,j}$ to $a_{(i \bmod n)+1,j}$;
- I: replace the permutation read left to right in each row with its inverse.
- C: replace the permutation read top to bottom in each column with its inverse.

Inverse of a permutation $p_1, p_2, \ldots, p_n$ is a permutation $q_1, q_2, \ldots, q_n$, such that $p_{q_i} = i$ for every $1 \leq i \leq n$.
One can see that after any sequence of operations every row and every column of the matrix will still be a permutation of $1, 2, \ldots, n$.

Given the initial matrix description, you should process m operations and output the final matrix.

Input

The first line contains a single integer t ($1 \leq t \leq 1000$) — number of test cases. t test case descriptions follow.

The first line of each test case description contains two integers n and m ($1 \leq n \leq 1000, 1 \leq m \leq 10^5$) — size of the matrix and number of operations.

Each of the next n lines contains n integers separated by single spaces — description of the matrix a ($1 \leq a_{i,j} \leq n$).

The last line of the description contains a string of m characters describing the operations in order, according to the format above.

The sum of n does not exceed 1000, and the sum of m does not exceed 10^5.

Output

For each test case, print n lines with n integers each — the final matrix after m operations.

Example

input

5
3 2
1 2 3
2 3 1
3 1 2
DR
3 2
1 2 3
2 3 1
3 1 2
LU
3 1
1 2 3
2 3 1
3 1 2
I
3 1
1 2 3
2 3 1
3 1 2
C

```
3 16
1 2 3
2 3 1
3 1 2
LDICRUCILDICRUCI
```

output

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

3 1 2
1 2 3
2 3 1

1 2 3
3 1 2
2 3 1

1 3 2
2 1 3
3 2 1

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

Note
Line breaks between sample test case answers are only for clarity, and don't have to be printed.

D. Flip and Reverse

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a string s of 0's and 1's. You are allowed to perform the following operation:

- choose a non-empty contiguous substring of s that contains an equal number of 0's and 1's;
- flip all characters in the substring, that is, replace all 0's with 1's, and vice versa;
- reverse the substring.

For example, consider s = 00111011, and the following operation:

- Choose the first six characters as the substring to act upon: **001110**11. Note that the number of 0's and 1's are equal, so this is a legal choice. Choosing substrings 0, 110, or the entire string would not be possible.
- Flip all characters in the substring: **110001**11.
- Reverse the substring: **100011**11.

Find the lexicographically smallest string that can be obtained from s after zero or more operations.

Input
The first line contains a single integer T ($1 \leq T \leq 5 \cdot 10^5$) — the number of test cases. Each of the following T lines contains a single non-empty string — the input string s for the respective test case.

All strings consist of characters 0 and 1, and their total length does not exceed $5 \cdot 10^5$.

Output
For each test case, on a separate line print the lexicographically smallest string that can be obtained from s after zero or more operations.

Example

input

```
3
100101
1100011
10101010
```

output

```
010110
0110110
10101010
```

Note
In the first test case a single operation should be applied to the entire string.

In the second test case two operations are needed: **011100**1, 011**0110**.

In the third test case the string stays the same after any operation.

E. Nim Shortcuts

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

After your debut mobile game "Nim" blew up, you decided to make a sequel called "Nim 2". This game will expand on the trusted Nim game formula, adding the much awaited second heap!

In the game, there are two heaps, each containing a non-negative number of stones. Two players make moves in turn. On their turn, a player can take any positive number of stones from either one of the heaps. A player who is unable to move loses the game.

To make the game easier to playtest, you've introduced developer shortcuts. There are n shortcut positions $(x_1, y_1), \ldots, (x_n, y_n)$. These change the game as follows: suppose that before a player's turn the first and second heap contain x and y stones respectively. If the pair (x, y) is equal to one of the pairs (x_i, y_i), then the player about to move loses instantly, otherwise they are able to make moves as normal. Note that in the above explanation the two heaps and all pairs are **ordered**, that is, x must refer to the size of the first heap, and y must refer to the size of the second heap.

The game release was followed by too much celebration, and next thing you know is developer shortcuts made their way to the next official update of the game! Players now complain that the AI opponent has become unbeatable at certain stages of the game. You now have to write a program to figure out which of the given initial positions can be won by the starting player, assuming both players act optimally.

Input

The first line contains two integers n and m ($1 \le n, m \le 10^5$) — the number of shortcut positions, and the number of initial positions that need to be evaluated.

The following n lines describe shortcut positions. The i-th of these lines contains two integers x_i, y_i ($0 \le x_i, y_i \le 10^9$). It is guaranteed that all shortcut positions are distinct.

The following m lines describe initial positions. The i-th of these lines contains two integers a_i, b_i ($0 \le a_i, b_i \le 10^9$) — the number of stones in the first and second heap respectively. It is guaranteed that all initial positions are distinct. However, initial positions are not necessarily distinct from shortcut positions.

Output

For each initial position, on a separate line print "`WIN`" if the starting player is able to win from this position, and "`LOSE`" otherwise.

Example

input
3 5 3 0 0 1 2 2 0 0 1 1 2 2 3 3 5 4
output
LOSE WIN LOSE WIN LOSE

F. Range Diameter Sum

time limit per test: 7 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a tree with n vertices numbered $1, \ldots, n$. A tree is a connected simple graph without cycles.

Let $\mathrm{dist}(u, v)$ be the number of edges in the unique simple path connecting vertices u and v.

Let $\mathrm{diam}(l, r) = \max \mathrm{dist}(u, v)$ over all pairs u, v such that $l \le u, v \le r$.

Compute $\sum_{1 \le l \le r \le n} \mathrm{diam}(l, r)$.

Input

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

The next $n - 1$ lines describe the tree edges. Each of these lines contains two integers u, v ($1 \le u, v \le n$) — endpoint indices of the respective tree edge. It is guaranteed that the edge list indeed describes a tree.

Output

Print a single integer — $\sum_{1 \le l \le r \le n} \operatorname{diam}(l, r)$.

Examples

input
4 1 2 2 4 3 2
output
10

input
10 1 8 2 9 5 6 4 8 4 2 7 9 3 6 10 4 3 9
output
224