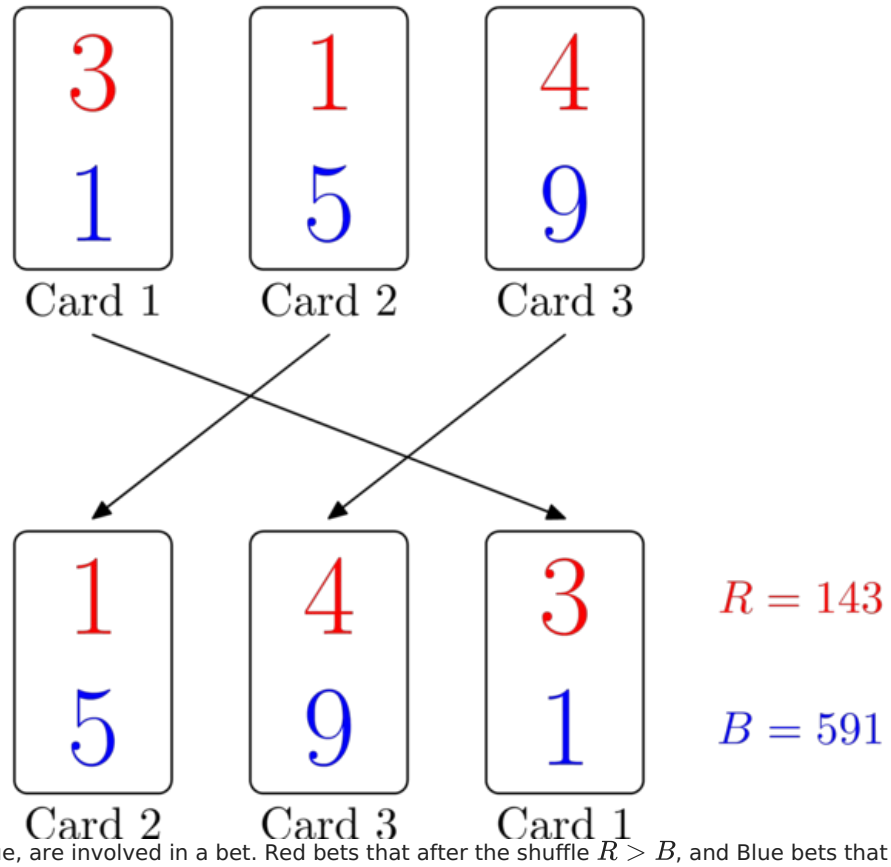
Codeforces Round #691 (Div. 2)

A. Red-Blue Shuffle

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

There are n cards numbered $1, \ldots, n$. The card i has a red digit r_i and a blue digit b_i written on it.

We arrange all n cards in random order from left to right, with all permutations of $1, \ldots, n$ having the same probability. We then read all red digits on the cards from left to right, and obtain an integer R. In the same way, we read all blue digits and obtain an integer B. When reading a number, leading zeros can be ignored. If all digits in a number are zeros, then the number is equal to 0. Below is an illustration of a possible rearrangement of three cards, and how R and B can be found.



Two players, Red and Blue, are involved in a bet. Red bets that after the shuffle $R > B$, and Blue bets that $R < B$. If in the end $R = B$, the bet results in a draw, and neither player wins.

Determine, which of the two players is more likely (has higher probability) to win the bet, or that their chances are equal. Refer to the Note section for a formal discussion of comparing probabilities.

Input

The first line contains a single integer T ($1 \le T \le 100$) — the number of test cases.

Descriptions of T test cases follow. Each test case description starts with a line containing a single integer n ($1 \le n \le 1000$) — the number of cards.

The following line contains a string of n digits $r_1, \ldots, r_n$ — red digits on cards $1, \ldots, n$ respectively.

The following line contains a string of n digits $b_1, \ldots, b_n$ — blue digits on cards $1, \ldots, n$ respectively.

Note that digits in the same line are not separated with any delimiters.

Output

Print T answers for the test cases in order, one per line.

If Red has a **strictly** higher change to win, print "RED".

If Blue has a **strictly** higher change to win, print "BLUE".

If both players are equally likely to win, print "EQUAL".

Note that all answers are **case-sensitive**.

Example

input
3 3 777 111 3 314 159 5 09281 09281
output
RED BLUE EQUAL

Note

Formally, let n_R be the number of permutations of cards $1, \ldots, n$ such that the resulting numbers R and B satisfy $R > B$. Similarly, let n_B be the number of permutations such that $R < B$. If $n_R > n_B$, you should print "RED". If $n_R < n_B$, you should print "BLUE". If $n_R = n_B$, print "EQUAL".

In the first sample case, $R = 777$ and $B = 111$ regardless of the card order, thus Red always wins.

In the second sample case, there are two card orders when Red wins, and four card orders when Blue wins:

- order $1, 2, 3$: $314 > 159$;
- order $1, 3, 2$: $341 > 195$;
- order $2, 1, 3$: $134 < 519$;
- order $2, 3, 1$: $143 < 591$;
- order $3, 1, 2$: $431 < 915$;
- order $3, 2, 1$: $413 < 951$.

Since $R < B$ is more frequent, the answer is "BLUE".

In the third sample case, $R = B$ regardless of the card order, thus the bet is always a draw, and both Red and Blue have zero chance to win.

B. Move and Turn

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

A robot is standing at the origin of the infinite two-dimensional plane. Each second the robot moves exactly 1 meter in one of the four cardinal directions: north, south, west, and east. For the first step the robot **can choose any of the four directions**, but then at the end of every second it **has to turn** 90 degrees left or right with respect to the direction it just moved in. For example, if the robot has just moved north or south, the next step it takes has to be either west or east, and vice versa.

The robot makes **exactly** n steps from its starting position according to the rules above. How many different points can the robot arrive to at the end? The final orientation of the robot can be ignored.

Input

The only line contains a single integer n ($1 \leq n \leq 1000$) — the number of steps the robot makes.

Output

Print a single integer — the number of different possible locations after **exactly** n steps.

Examples

input
1
output
4

input
2
output
4

input
3

output
12

Note
In the first sample case, the robot will end up 1 meter north, south, west, or east depending on its initial direction.

In the second sample case, the robot will always end up $\sqrt{2}$ meters north-west, north-east, south-west, or south-east.

C. 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

D. 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.

E. 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.

F. 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.