Problem A. Alice and Bob

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

This is an unusual problem. You should write a program that works as both Alice and Bob. It will be executed twice, independently: once for Alice and once for Bob. The output of Alice's execution will be an input for Bob's execution.

Alice has a binary string s of length n. She wants to send it to Bob. But, unfortunately, she got some constraints on what she can send.

Besides the string s, Alice also receives a pattern p of length m. Exactly n characters in this pattern are "0" and / or "1", and all other are "?".

Alice has to output a string t of length m which matches pattern p. That is, for each position i in pattern p, if the character p_i on this position is a digit, there must be $t_i = p_i$.

Bob receives a string t from Alice. He must output an initial Alice's string s.

Input

For Alice's execution, the first line contains a string "Alice", the second line contains two integers n and m, the third line contains a string s of length n, and the fourth line contains a pattern p of length m.

For Bob's execution, the first line contains a string "Bob", the second line contains two integers n and m, and the third line contains a string t of length m.

In all tests except samples, $1 \le n \le 1000$ and $m = 2n + 42$.

Output

For Alice's execution, output a string t of length m which matches pattern p.

For Bob's execution, output the initial Alice's string s of length n.

Examples

standard input	standard output
Alice 4 13 0011 00??11???????	0011111100001
Bob 4 13 0011111100001	0011

Problem B. Range Diameter Sum

Input file:	*standard input*
Output file:	*standard output*
Time limit:	10 seconds
Memory limit:	512 mebibytes

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

Let $\text{dist}(u, v)$ be the number of edges in the unique simple path connecting vertices u and v.
Let $\text{diam}(l, r) = \max \text{dist}(u, v)$ over all pairs u, v such that $l \le u, v \le r$.

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

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 — $\sum_{1 \le l \le r \le n} \text{diam}(l, r)$.

Examples

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

Problem C. Flip and Reverse

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

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: **`00111011`**. 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: **`11000111`**.
- Reverse the substring: **`10001111`**.

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 ($1 \le T \le 5 \cdot 10^5$) — 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 are non-empty, 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

standard input	standard output
3 100101 1100011 10101010	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: **`0111001`**, `011`**`0110`**.

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

Problem D. Free Throws

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

In basketball, players make throws from close distance or long distance, which bring 2 and 3 points respectively. If a player A illegally attacks a player B from the opposing team when the latter is attempting an x-point throw, then the throw is nullified, and the player B performs x *free throws* as a penalty. Each successful free throw gives the team of player B one point. The player A receives a personal warning. After sixth personal warning the player is removed from the game. It is impossible to attack a player during their free throw.

In modern basketball, analysts know almost everything about players, so a good analyst can easily tell the probability to hit the goal for each player when he is throwing a 3-pointer, a 2-pointer, or a free throw. The probability is measured as integer number of percents.

...Once in a well-known league the roster of a well-known team before the game was too short due to medical restrictions. Then the head coach decided to let the head analyst of the team to play. But being an expert in digital basketball, in real game the analyst has only one working skill — to illegally attack players of opposing team and collect personal warnings.

Right before the game the head analyst built a statistical model of the game — all throws of opposing team in order, along with the type of each throw (2 or 3 points). He plans to choose an optimal way to act under the limit of 6 possible personal warnings to reduce expected total score of opposing team. An illegal attack does not affect any throws other than the one it's performed upon.

Your task is to calculate the minimum possible expected score over all possible illegal attack strategies.

Input

The first line contains two integers n and t ($10 \le n \le 20$, $1 \le t \le 1000$) — the number of players in the opposing team, and the number of throws in the model respectively.

The following n lines give the statistical data of those players. The i-th of those lines contains three integers $P_{i,3}$, $P_{i,2}$ and $P_{i,1}$ ($0 \le P_{i,j} \le 100$) — the probability (in percents) for the player i to score a 3-point throw, a 2-point throw, and a free throw respectively.

The following t lines describe the throws in the model in order they happen. The j-th lines contains two integers p_j, s_j ($1 \le p_i \le n$, $2 \le s_j \le 3$) — the number of the player who makes the throw, and the value of the throw j.

Output

Print one real number — smallest possible expected score which can be achieved if the analyst acts optimally, with absolute or relative error 10^{-9} or better.

Example

standard input	standard output
10 15 87 84 87 78 85 90 70 90 92 82 82 87 78 89 91 91 85 89 78 90 93 88 87 88 89 91 92 85 81 93 10 2 3 2 1 3 5 3 1 3 7 3 4 3 6 2 1 3 2 2 1 2 7 3 1 3 8 3 9 2	32.88

Problem E. Row GCD

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

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 $\mathrm{GCD}(a_1 + b_j, \ldots, a_n + b_j)$.

Example

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

Problem F. Interactive Valuer

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

This is an interactive problem.

In an IOI-style contest, the scoring system of a problem usually involves several *subtasks*. Each subtask is described by the score value awarded for solving the subtask, and a list of test cases a submission needs to pass. Note that the test lists for different subtasks may or may not intersect.

Additionally, *dependencies* may be defined for a subtask as a list of other required subtasks. A subtask is considered to be *solved* if both conditions hold:

- all test cases in this subtask are passed,
- all test cases needed for all required subtasks to be solved are passed.

Note that the definition is effectively recurisve, and in general there may be cyclic dependencies between subtasks.

Resulting score of a submission is the sum of scores for all solved subtasks.

Under IOI-style scoring, sometimes it is possible to stop running tests early, because the verdict for the remaining tests will not affect the score. It may reduce load on the invokers, which is important for the contests with big number of participants.

One of possible ways to do that is so-called *interactive valuer*. For example, this idea was implemented in ejudge Contest Control System by Alexander Chernov.

The valuer program reads information about subtasks. Then the testing process starts with current test case index C set to zero.

At any time valuer may do one of two actions:

- Stop the testing and return the submission score.
- Increase C by any integer $X > 0$, and ask the system to invoke the solution on a test case with the index equal to the new value of C. The testing system communicates the result of evaluation to the valuer: 1 if the test case was passed and 0 otherwise.

If the valuer asks the system to run a test case with an illegal index, evaluating process is immediately failed.

Your task is to implement the interactive valuer. It should correctly evaluate the submission while running the solution on the smallest possible number of tests.

Interaction Protocol

The interaction is started by the jury program by sending the subtask information. The first line contains two integers N and T ($1 \le N \le 100$, $1 \le T \le 256$) — the number of subtasks, and the total number of test cases in the problem.

The following N lines describe subtask test lists, one per line. Each subtask description starts with integer s_i ($1 \le s_i \le 100$) — the score for the subtask i, followed by integer t_i ($1 \le t_i \le T$) — the number of test cases in that subtask, followed by t_i distinct integers representing test cases included in that subtask. Test cases are numbered with sequential integers between 1 and T. You may assume that the sum of all scores s_i is equal to 100.

The following N lines describe dependencies. The i-th line describes dependencies of the subtask i and starts with the number of required subtasks d_i ($0 \le d_i \le N$), followed by d_i pairwise distinct integers between 1 and N (inclusive) — indices of required subtasks. Note that there are no restrictions on possible loops or even self-loops in dependencies graph.

Then the interaction starts. Initially, $C = 0$. Your solution should make queries to the jury program. Each query can be made by printing a single integer X ($-256 \le X \le 100$).

If $X < 0$, then C will be increased by $-X$, and the test case C will be invoked. Then, your program should read a single integer: 1 if the test case C is passed, and 0 otherwise.

If $X \geq 0$, the interaction stops and the final submssion score is returned as X.

If your program asks to invoke a test case C that does not exist, or when the jury program can conclude that your program will not be able to meet the optimality requirement, the interaction stops and your solution receives the Wrong Answer verdict.

Example

standard input	standard output
3 8 21 4 1 2 5 3 39 3 3 6 7 40 1 8 0 1 2 1 1	-1
1	-1
1	-1
1	-2
0	-1
1	-1
1	39

Note

Don't forget to flush your output after each query, otherwise verdicts Time Limit Exceeded or Idleness Limit Exceeded may occur.

Problem G. Latin Square

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

You are given a square matrix of size n. Every row and every column of this matrix is a permutation of 1, 2, ..., n. Let $a_{i,j}$ be the element at the intersection of i-th row and j-th column for every $1 \le i, j \le n$. 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, ..., p_n is a permutation q_1, q_2, ..., q_n, such that $p_{q_i} = i$ for every $1 \le i \le n$.

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 ($1 \le t \le 1000$) — number of test cases. t test case descriptions follow.

The first line of each test case description contains two integers n and m ($1 \le n \le 1000, 1 \le m \le 10^5$) — 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 ($1 \le a_{i,j} \le n$).

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

standard input	standard output
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	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.

Problem H. Submatrices of given rank

Input file:	*standard input*
Output file:	*standard output*
Time limit:	5 seconds
Memory limit:	512 mebibytes

You are given a binary matrix A of size $n \times m$ and integer k. Find the number of its non-empty submatrices B such that rank B over $\mathbb{F}_2$ is exactly k.

Submatrix of M is a matrix consisting of elements at the intersection of some consecutive rows of M and some consecutive columns of M.

Rank of the matrix M over $\mathbb{F}_2$ is the maximum cardinality of independent set of rows of M. Set of rows S is called *independent*, if for any non-empty subset $T = \{t_1, \ldots, t_p\} \subseteq S$, sum $t_1 + \cdots + t_p$ has at least one non-zero coordinate. Sum of two rows $(a_1, ..., a_q)$ and $(b_1, ..., b_q)$ over $\mathbb{F}_2$ is row $(a_1 \oplus b_1, ..., a_q \oplus b_q)$, where $\oplus$ is exclusive or.

Input

The first line contains three integers n, m and k ($1 \le n, m, k \le 500$) — the number of rows and columns of the matrix A and required rank.

The following n lines contain m characters, each of which is 0 or 1. If the j-th character in the i-th line is 0, then $A_{ij} = 0$, otherwise $A_{ij} = 1$.

Output

Print one integer: the number of non-empty submatrices with rank k over $\mathbb{F}_2$.

Examples

standard input	standard output
3 3 2 100 011 001	6
7 10 4 0011000100 1010010101 0011010111 1111100100 0110001000 1011101101 0001100000	131

Problem I. Nim Shortcuts

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

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

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

Problem J. Glass Half Spilled

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

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 — the number of glasses ($1 \le n \le 100$).

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

standard input	standard output
3 6 5 6 5 10 2	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.

Problem K. Red-Blue Shuffle

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

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.

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

standard input	standard output
3 3 777 111 3 314 159 5 09281 09281	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.

Problem L. Truth Table

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	512 mebibytes

Truth table is a straightforward way of representing boolean functions. A truth table for a function $f(x_1, \ldots, x_n)$ is a matrix with 2^n rows and $n + 1$ columns, where each of the first n columns correspond to the value of some variable and the last column contains the value of the function when variables have these certain values.

Each combination of variables must be present in the truth table exactly once, and combinations must be in lexicographical order. So, the first row of the truth table starts with $000\ldots00$, the second — with $000\ldots01$, the last — with $111\ldots11$.

We can extend a truth table to represent more than one function. Such extended truth table contains n columns for variables and one more for each function. For example, here is a table for functions $x_1 \wedge x_2$ and $x_1 \to x_2$ (header is for clarity):

x_1	x_2	$x_1 \wedge x_2$	$x_1 \to x_2$
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	1

We prepared for you an extended truth table for k functions of n variables (that is, a matrix 2^n by $n + k$). Unfortunately, it was not properly wrapped and transportation conditions were very harsh, so its rows and columns became randomly shuffled. Next time we will spend a penny on insurance, but for now it is your task to fix it.

Find a permutation $p_1, \ldots, p_{2^n}$ of rows and a permutation $q_1, \ldots, q_{n+k}$ of columns and such that after permuting rows and columns accordingly the table becomes a truth table.

Formally, let the shuffled table be $\{a_{ij}\}$. Then the following should hold:

- $a_{p_1 q_1} a_{p_1 q_2} \ldots a_{p_1 q_n} = 00\ldots0$
- $a_{p_2 q_1} a_{p_2 q_2} \ldots a_{p_2 q_n} = 00\ldots1$
- $\ldots$
- $a_{p_{2^n} q_1} a_{p_{2^n} q_2} \ldots a_{p_{2^n} q_n} = 11\ldots1$

Input

In the first line of input there are two numbers n, k ($n + k \le 23$, $1 \le n \le 20$, $k \ge 0$). Each of the next 2^n lines contains a binary string of length $n + k$, denoting the corresponding row of the table.

Output

If it is possible to restore the table, print "`Yes`" in the first line. Then print the permutations p and q as defined in the statement. If there are multiple answers, print any of them.

If there is no answer, print a single word "`No`".

Example

standard input	standard output
2 1 101 100 010 111	Yes 2 1 3 4 2 3 1

Problem M. Expected area

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	512 mebibytes

Given two collections of segments $S = \{(l_1, r_1), \dots (l_n, r_n)\}$ and $T = \{(s_1, t_1), \dots (s_n, t_n)\}$ in the real line, find the expected area of $\bigcup_{i=1}^{n} \left([l_i, r_i] \times \left[s_{p(i)}, t_{p(i)}\right]\right)$, where p is taken equiprobably over all permutations on n elements. Here, $[a, b] \times [c, d]$ is a rectangle in the two-dimensional plane defined as $\{(x, y) \mid a \le x \le b \text{ and } c \le y \le d\}$.

It can be shown that the answer can be written in the form $\frac{p}{q}$ where p and q are coprime integers and $q \not\equiv 0 \bmod 998\,244\,353$. The answer modulo $998\,244\,353$ is equal to $(p \cdot q^{-1}) \bmod 998\,244\,353$.

Input

The first line contains integer n ($2 \le n \le 2 \cdot 10^5$) — the number of segments in each set.

Each of the next n lines contains two integers l_i and r_i ($0 \le l_i < r_i < 998\,244\,353$) — endpoints of the i-th segment of the collection S.

Each of the next n lines contains two integers s_i and t_i ($0 \le s_i < t_i < 998\,244\,353$) — endpoints of the i-th segment of the collection T.

Output

Print one integer: the expected area of union of rectangles modulo $998\,244\,353$.

Examples

standard input	standard output
2 0 1 2 3 0 1 1 3	3
5 3 10 1 6 1 8 1 9 5 7 5 9 0 9 1 9 2 10 2 4	99824516