Problem A. Airport Assignments

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 seconds
Memory limit:	1024 mebibytes

An airport has n_1 aircraft waiting for service, numbered from 1 to n_1, and n_2 gates arranged from left to right, numbered from 1 to n_2. There are m compatible aircraft–gate pairs. A pair (u, v) means that aircraft u can be assigned to gate v.

For every contiguous gate zone $[\ell, r]$ (where $1 \le \ell \le r \le n_2$), the airport operations team considers assignments subject to the following rules:

- all aircraft $1, 2, \ldots, n_1$ are available;
- only gates with numbers in $[\ell, r]$ may be used;
- an aircraft may be assigned only to a compatible gate.

Each aircraft can be assigned to at most one gate, and each gate can serve at most one aircraft. Let $f(\ell, r)$ be the maximum number of aircraft that can be assigned simultaneously within gate zone $[\ell, r]$.

To generate the airport's audit checksum, compute

$$\sum_{1 \le \ell \le r \le n_2} f(\ell, r) \cdot \ell \cdot r \cdot ((\ell \oplus r) + 1),$$

where $\oplus$ denotes the bitwise XOR operation.

Output the value of this checksum modulo 998 244 353.

Input

The first line contains a single integer t ($1 \le t \le 2000$), the number of test cases. For each test case:

The first line contains three integers n_1, n_2, and m ($1 \le n_1, n_2 \le 5000$, $1 \le m \le 10^4$), representing the number of aircraft, the number of gates, and the number of compatible aircraft–gate pairs.

Each of the next m lines contains two integers u and v ($1 \le u \le n_1$, $1 \le v \le n_2$), meaning that aircraft u can be assigned to gate v.

No compatible pair is listed more than once. The sum of n_1 does not exceed 5000. The sum of n_2 also does not exceed 5000.

Output

For each test case, output a single line containing one integer: the required audit checksum modulo 998 244 353.

Example

standard input	standard output
4 3 3 5 1 1 2 1 2 2 2 3 3 3 3 4 6 1 2 1 4 2 3 2 1 3 4 3 3 2 2 1 1 2 4 3 5 1 3 2 1 3 3 4 2 4 1	81 529 12 81

Problem B. Best Coupon Tour

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

There are n cities, numbered from 1 to n. There are also $n-1$ bidirectional roads connecting these cities, so that for every two cities x and y, there exists a path from x to y. The i-th road connects cities u_i and v_i, and has a cost c_i. Every time you pass through road i, you spend c_i coins.

You are now at city 1 and want to travel to visit all cities. Every second, if you are at city x, you can choose a city y that is directly connected with city x by a road, move to city y, and pay the cost of the respective road. Each road can be passed **at most twice**. When you have visited all cities and are at city 1, you can finish your travel. The cost of the travel is the total cost of all the moves you made.

There is also an integer k. During your travel, you can use a free coupon once, making your following k moves have no cost.

For every k from 0 to $2n-2$, you should calculate the minimum cost of a travel that visits all cities and uses the coupon optimally.

Input

The first line contains a single integer t ($1 \le t \le 400$), the number of test cases. For each test case:

The first line contains an integer n ($1 \le n \le 2000$), denoting the number of cities.

The next $n-1$ lines describe the roads. The i-th of these lines contains three integers, u_i, v_i, and c_i ($1 \le u_i, v_i \le n$, $0 \le c_i \le 10^9$), denoting the cities connected by the i-th road and the cost of that road.

There is a path between any pair of cities. The sum of n does not exceed 2000.

Output

For each test case, output one line containing $2n-1$ integers: the answers for k from 0 to $2n-2$.

Example

standard input	standard output
2 5 1 2 2 2 3 3 2 4 1 4 5 3 7 1 2 1 1 3 1 1 4 4 3 5 5 3 6 1 6 7 4	18 15 12 10 8 5 3 2 0 32 27 22 21 17 13 12 8 4 3 2 1 0

Note

In the first test case, we can always travel along the path $1,2,3,2,4,5,4,2,1$, with the cost sequence $2,3,3,1,3,3,1,2$. The cost-free segments for each k are shown below.

k	cost-free segment	answer
0	$[]$	18
1	$[3]$	15
2	$[3, 3]$	12
3	$[2, 3, 3]$	10
4	$[3, 3, 1, 3]$	8
5	$[3, 3, 1, 3, 3]$	5
6	$[2, 3, 3, 1, 3, 3]$	3
7	$[2, 3, 3, 1, 3, 3, 1]$	2
8	$[2, 3, 3, 1, 3, 3, 1, 2]$	0

Problem C. Counting Stone Configurations

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

You are given n piles of stones, labeled by integers from 1 to n. The i-th pile initially contains a_i stones. You also have a bag, which is initially empty.

You may perform the following operations, any number of times, in any order:

- Choose a pile that is not empty, remove **one** stone from this pile, and put it into the bag.
- Choose a pile (possibly empty), remove **exactly** n stones from the bag, and put all these n stones into the chosen pile. This operation can be performed only when the bag contains at least n stones.

You may stop performing operations at any time. In the end, the bag may contain any (possibly nonzero) number of stones.

Determine how many distinct final configurations of the n piles can be obtained from the initial configuration by performing a finite sequence of operations. Output the answer modulo $998\,244\,353$.

Two final configurations are considered different if there exists some index i such that the number of stones in pile i differs between them. Recall that the piles are labeled and their order matters.

Input

The first line contains a single integer n ($1 \le n \le 3000$), the number of piles.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$), where a_i is the initial number of stones in the i-th pile.

Output

Output a single integer: the number of distinct final configurations of the piles that can be obtained, modulo $998\,244\,353$.

Examples

standard input	*standard output*
2 2 5	36
3 1 2 2	54
8 5 1 4 2 0 3 1 4	2170629

Problem D. Digital Signal Cascade

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 seconds
Memory limit:	1024 mebibytes

A digital signal-processing line consists of n modules arranged in a chain and numbered from 1 to n. Module i initially stores a signed integer signal level a_i.

Every module must be recalibrated exactly once, but you may choose the recalibration order:

- Choose a permutation $p_1, p_2, \ldots, p_n$ of $1, 2, \ldots, n$.
- Then, for i from 1 to n in the natural order, recalibrate module p_i by executing

  $$a_{p_i} \leftarrow a_{p_i} + a_{p_i-1} + a_{p_i+1},$$

  where $a_0 = 0$ and $a_{n+1} = 0$ represent the missing neighbors beyond the ends of the chain.

Every update takes effect immediately, so a module recalibrated later uses the current signal levels, including any changes made earlier.

Your goal is to choose the recalibration order that maximizes the sum of all signal levels after every module has been recalibrated.

Input

The first line contains a single integer t ($1 \le t \le 10^4$), the number of test cases. For each test case:

The first line contains an integer n ($1 \le n \le 2 \cdot 10^5$), the number of signal modules.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-10^8 \le a_i \le 10^8$), the initial signed signal levels.

The sum of n does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single line containing the maximum possible sum of all signal levels after the recalibration cascade.

Example

standard input	*standard output*
4 3 1 2 3 4 -1 3 7 6 5 4 1 0 5 2 6 4 -3 7 5 -9 3	22 85 72 79

Problem E. Edge Resource Allocation

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	1024 mebibytes

An edge-computing network consists of n processing nodes, numbered from 1 to n, and n service zones, also numbered from 1 to n. There are m service links. A link (u, v) means that processing node u can serve zone v.

You will be asked q independent scheduling queries. In each query, you are given three integers (a, b, k).

For a fixed query (a, b, k), consider one normalized scheduling period. Initially, the useful-work counter of every processing node and the overhead counter of every service zone are equal to 0. You may perform the following operation any number of times:

- Choose an arbitrary subset S of processing nodes (possibly empty), and choose a positive real duration $p > 0$.
- Let $N(S)$ be the set of service zones linked to at least one processing node in S.
- Increase the useful-work counter of every processing node in S by $a \cdot p$.
- Increase the overhead counter of every service zone in $N(S)$ by $b \cdot p$.

Let $p_1, p_2, \ldots, p_\ell$ be the durations used in all operations. They must satisfy

$$\sum_{i=1}^{\ell} p_i \le 1.$$

The total system load is the sum of all useful-work counters and all service-zone overhead counters. You must keep this total load at most k. Subject to this constraint, maximize the total useful work completed by all processing nodes.

For every query (a, b, k), the counters are reset to 0. Compute the maximum possible total useful work for that query.

Input

The first line contains a single integer t ($1 \le t \le 1000$), the number of test cases. For each test case:

The first line contains three integers n, m, and q ($1 \le n \le 2000$, $0 \le m \le 10^4$, $1 \le q \le 2 \cdot 10^5$), representing the number of processing nodes and service zones, the number of service links, and the number of queries.

Each of the next m lines contains two integers u and v ($1 \le u, v \le n$), meaning that processing node u can serve zone v.

Each of the next q lines contains three integers a, b, and k ($0 \le a, b \le 10^6$, $0 \le k \le 10^9$), describing the useful-work rate, the service-zone overhead rate, and the total load budget for a query.

No service link is listed more than once. The sum of n does not exceed 2000. The sum of m does not exceed 10^4. The sum of q does not exceed $2 \cdot 10^5$.

Output

For each query, output a single real number: the maximum possible total useful work completed by the processing nodes. Let your answer be x and the jury's answer be y. Your answer will be considered correct if $\frac{|x-y|}{\max(1,|y|)} \le 10^{-6}$.

Example

standard input	*standard output*
2 5 8 3 1 2 1 3 2 4 2 5 3 1 3 3 4 2 5 4 1 3 5 2 1 2 5 2 3 2 3 3 1 2 1 1 2 1 1 0 2 0 2 2 3 1 4	1.250000000000000 1.333333333333333 2.142857142857144 2.000000000000000 0.000000000000000 3.000000000000000

Problem F. Facing Floodlights

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 mebibytes

A straight road is divided into n consecutive sections, numbered from 1 to n. Directional floodlights can be installed along the road. Each floodlight is characterized by its position x (an integer between 1 and n) and the direction it faces: left or right.

All floodlights share the same beam range L, which is a non-negative integer.

- A floodlight at position x facing right illuminates all road sections with indices in the segment $[x, x + L]$.
- A floodlight at position x facing left illuminates all road sections with indices in the segment $[x-L, x]$.

Only road sections with indices between 1 and n exist; any part of a beam outside this range is ignored.

Initially, there are no floodlights. You need to process q operations of two types:

- **Install a floodlight**. An operation of the form "`1` d x" installs a new floodlight at position x.

  If $d = 0$, the floodlight faces left.

  If $d = 1$, the floodlight faces right.

  Multiple floodlights may be installed at the same position. Operations are given in chronological order, and no floodlight is ever removed.

- **Illumination query**. An operation of the form "`2` ℓ r" asks the following question:

  Consider all floodlights installed so far. Suppose they all have the same beam range L. Find the smallest integer $L \geq 0$ such that every road section with index in $[\ell, r]$ is illuminated by at least one floodlight. Floodlights outside $[\ell, r]$ may help illuminate sections inside it. If no integer L can illuminate every section in $[\ell, r]$, no matter how large L is, output -1 for this query.

Input

The first line contains two integers n and q ($1 \leq n, q \leq 3 \cdot 10^5$), representing the number of road sections and the number of operations.

Each of the next q lines describes one operation in the order in which it must be processed:

- "`1` d x": install a floodlight at position x facing direction d ($d \in \{0, 1\}$, $1 \leq x \leq n$). Here, $d = 0$ means left and $d = 1$ means right.
- "`2` ℓ r": query the minimum beam range L for the segment $[\ell, r]$ ($1 \leq \ell \leq r \leq n$).

Output

For each query of the form "`2` ℓ r", output a single line containing one integer:

- the minimum integer $L \geq 0$ such that every road section in $[\ell, r]$ is illuminated, or
- -1 if this is impossible.

Example

standard input	*standard output*
10 11 1 0 6 2 5 8 1 1 3 2 9 10 2 3 9 1 1 8 2 1 9 2 1 8 1 0 4 2 3 7 2 2 9	-1 7 6 5 5 4 4

Problem G. Gapped Copies

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

You are given a string s of length n.

For each position i where $1 \le i \le n$, let t_i be the string obtained from s by deleting the i-th character of s (and keeping the relative order of all other characters).

For two strings x and y, let $\operatorname{lcp}(x, y)$ denote the *length* of their longest common prefix, that is, the largest integer $\ell \ge 0$ such that the first ℓ characters of x and y are equal (if $x_1 = y_1$, $x_2 = y_2$, …, $x_\ell = y_\ell$).

Your task is to choose two **different** positions i and j such that the value $\operatorname{lcp}(t_i, t_j)$ is as large as possible, and compute this maximum possible value.

Formally, you need to find

$$\max_{1 \le i < j \le n} \operatorname{lcp}(t_i, t_j).$$

You do not need to output i or j, only the maximum value of $\operatorname{lcp}(t_i, t_j)$.

Input

The first line contains a single integer t ($1 \le t \le 2 \cdot 10^5$), the number of test cases. For each test case:

A single line contains a string s that consists of lowercase English letters ($2 \le |s| \le 5 \cdot 10^5$).

The sum of $|s|$ does not exceed $5 \cdot 10^5$.

Output

For each test case, output a single integer on a separate line, representing the maximum possible value of $\operatorname{lcp}(t_i, t_j)$ over all pairs of indices $i \ne j$.

Example

standard input	*standard output*
5 cdce eaaaa dacda cacdd zz	2 4 3 4 1

Problem H. How Many Trees?

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	1024 mebibytes

You are given m depth-first search (DFS) orders of an unknown rooted tree.

We consider rooted trees with n vertices labeled from 1 to n, where vertex 1 is the root. The tree is undirected and connected, and has exactly $n-1$ edges.

DFS definition. For a fixed rooted tree, a DFS order is obtained as follows:

- All vertices are initially unvisited.
- Start at the root 1, mark it as visited and **output** it.
- When you are at some vertex v, consider all children of v in the rooted tree. Choose its children in **any** order. For each chosen child u that is still unvisited, go to u, mark it as visited, output u, and continue the DFS from u in the same way.

For a fixed tree, different choices of the order in which the children of each vertex are processed may produce different DFS orders. All DFS orders are permutations of $1, 2, \ldots, n$ and always start with 1.

You are given m permutations of $1, 2, \ldots, n$, each starting with 1. You are told that each of them is a DFS order of the **same** rooted tree (with root 1) in the sense defined above (possibly using different choices of child orders for different DFS runs).

Your task is to determine how many different rooted trees could produce **all** of these m DFS orders.

Two rooted trees are considered different if their sets of edges are different.

Because the answer can be very large, you should output it modulo $10^9 + 7$.

Input

The first line contains two integers n and m ($1 \le n, m \le 500$).

Each of the next m lines contains a permutation $p_{k,1}, p_{k,2}, \ldots, p_{k,n}$ of the integers $1, 2, \ldots, n$, describing the k-th DFS order.

For every k, all $p_{k,i}$ are distinct and $p_{k,1} = 1$.

Output

Output a single integer: the number of rooted trees with vertex set $1, 2, \ldots, n$ and root 1 for which **every** given sequence is a valid DFS order of that tree, taken modulo $10^9 + 7$.

Examples

standard input	*standard output*
3 2 1 2 3 1 3 2	1
4 1 1 2 3 4	5
5 2 1 2 3 4 5 1 2 4 3 5	3

Problem I. I Will Be the Champion

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	1024 mebibytes

You and your two teammates are participating in a programming contest, but due to the final exams, you all have to leave early. Each of you has a specific departure time ℓ_i (the time in minutes after the contest starts when you must leave). After the contest, you review the solutions and realize that each problem is perfectly suited for one of you: problem i is best solved by person p_i, who can complete it in c_i minutes without any wrong submissions. Unfortunately, due to the pressure of exams, you didn't perform well in the original contest.

In a dream, you see the champion team from the live broadcast: they solved all n problems with a total penalty time t. When you wake up, you discover you've been reborn on the morning of the contest day. Now, with perfect memory of all problem details and the champion's performance, you want to determine if there's a way to beat them this time.

The contest rules state:

- Only one computer is available, so problems must be solved sequentially (one after another).
- Each problem i must be assigned to person p_i, who must start solving it before their departure time ℓ_{p_i}.
- If person p_i starts solving problem i at time s, they must finish by $s + c_i \le \ell_{p_i}$.
- The total penalty time is the sum of the completion times for all solved problems.

Given n, t, and the parameters p_i and c_i for each problem, determine if there exists an order of solving the problems such that:

- All n problems are solved.
- The total penalty time is **strictly less** than t.

If such an arrangement exists, output "`YES`"; otherwise, output "`NO`".

Input

The first line contains a single integer q ($1 \le q \le 10^5$), the number of test cases. For each test case:

The first line contains four integers: n, ℓ_1, ℓ_2, and ℓ_3 ($1 \le n \le 10^5$, $1 \le \ell_1, \ell_2, \ell_3 \le 3 \cdot 10^7$), representing the number of problems and the departure times (in minutes after contest start) of the three teammates.

Each of the following n lines contains two integers, p_i and c_i ($1 \le p_i \le 3$, $1 \le c_i \le 300$), where p_i denotes the designated teammate for problem i, and c_i is the required time (in minutes) for them to complete it.

The last line contains a single integer t ($1 \le t \le 10^{13}$), representing the total penalty time of the champion team.

The sum of n over all test cases does not exceed 10^5.

Output

For each test case, if there exists an order of solving problems that satisfies the constraints (all problems solved, penalty strictly less than t), output "`YES`"; otherwise, output "`NO`".

Examples

standard input	*standard output*
2 3 100 150 175 1 100 2 25 3 50 401 5 100 200 300 1 30 1 30 1 40 2 110 3 50 1275	YES NO
1 1 100 300 300 1 300 300	NO

Problem J. Just the Right Shift

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	1024 mebibytes

You are given a sequence of n integers: $a_1, a_2, \ldots, a_n$.

Consider all n possible cyclic shifts (rotations) of this sequence. For a starting position p ($1 \le p \le n$), the corresponding cyclic shift is the sequence

$$b_1, b_2, \ldots, b_n = a_p, a_{p+1}, \ldots, a_n, a_1, a_2, \ldots, a_{p-1}.$$

For this shifted sequence, define its **prefix maximum sequence** $c_1, c_2, \ldots, c_n$ as

$$c_i = \max(b_1, b_2, \ldots, b_i) \text{ for all } i = 1, 2, \ldots, n.$$

We compare two sequences of the same length by the usual lexicographical order: c is lexicographically smaller than d if and only if there exists an index i such that $c_1 = d_1$, $c_2 = d_2$, …, $c_{i-1} = d_{i-1}$, and $c_i < d_i$.

Your task is to choose a cyclic shift of the original sequence such that its prefix maximum sequence is the lexicographically smallest among all n possible cyclic shifts.

Input

The first line contains a single integer t ($1 \le t \le 2 \cdot 10^5$), the number of test cases. For each test case:

The first line contains a single integer n ($1 \le n \le 5 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$).

The sum of n over all test cases does not exceed $5 \cdot 10^5$.

Output

For each test case, output one line containing n integers: the lexicographically smallest prefix maximum sequence after a cyclic shift.

Example

standard input	*standard output*
6 5 5 4 3 2 1 4 1 3 1 3 5 1 2 1 3 1 5 2 3 2 1 1 6 1 2 1 3 1 4 6 2 1 3 1 4 1	1 5 5 5 5 1 3 3 3 1 1 2 2 3 1 1 2 3 3 1 2 2 3 3 4 1 2 2 3 3 4

Problem K. Knot Equivalence

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	1024 mebibytes

You are given two integer sequences A and B of lengths $2n$ and $2m$, respectively. All elements are non-zero integers.

The sequence $A = (a_1, a_2, \ldots, a_{2n})$ satisfies the following condition: for every integer i such that $1 \le i \le n$, there are exactly two indices j such that $|a_j| = i$. In particular, $\{|a_j| : 1 \le j \le 2n\} = \{1, 2, \ldots, n\}$, and each absolute value appears exactly twice (if we disregard the signs).

The sequence $B = (b_1, b_2, \ldots, b_{2m})$ satisfies a similar condition: for every i such that $1 \le i \le m$, there are exactly two indices j such that $|b_j| = i$.

You may perform the following operations on the sequence A, in any order and any number of times:

1. **Cyclic shift.** Regard the current sequence as a cycle, and perform a rotation:

$$(a_1, a_2, \ldots, a_{2k}) \longrightarrow (a_{r+1}, a_{r+2}, \ldots, a_{2k}, a_1, \ldots, a_r)$$

   where $0 \le r < 2k$.

2. **Delete a prefix of a pair of opposite numbers.** If the current sequence has the form $A = (x, -x, S_1)$, where $x \ne 0$ and S_1 may be empty, you may replace A by S_1; that is, delete the first two elements, x and $-x$.

3. **Insert a new pair of opposite numbers at the front.** If the current sequence is $A = (S_1)$ (may be empty), you may replace it by $(x, -x, S_1)$, where $x \ne 0$, and neither x nor $-x$ appears anywhere in S_1.

4. **Reverse and negate.** If the current sequence is

$$A = (a_1, a_2, \ldots, a_{2k}),$$

   you may replace it by

$$\mathrm{nRev}(A) = (-a_{2k}, -a_{2k-1}, \ldots, -a_1).$$

5. **Rearrange across a cut.** Choose a cut that splits the current sequence into a prefix and a suffix. Suppose the sequence can be written as

$$A = (S_1, x, S_2, S_3, y, S_4)$$

   where $x > 0$, $y \in \{x, -x\}$, and the cut is between S_2 and S_3. Here, S_1, S_2, S_3, and S_4 may be empty sequences. Then you may apply one of the following transformations:

   - If $y = x$ (that is, you choose a pair x and x), you may replace A by

$$(S_1, \mathrm{nRev}(S_3), x, \mathrm{nRev}(S_4), S_2, y).$$

   - If $y = -x$ (that is, you choose a pair x and $-x$), you may replace A by

$$(S_1, S_4, x, S_3, S_2, y).$$

You have to determine if it is possible to transform sequence A into a sequence that is *equivalent* to sequence B by applying these operations a finite number of times.

Let $C = (c_1, \ldots, c_{2k})$ and $D = (d_1, \ldots, d_{2k})$ be two integer sequences of the same length. We say that C and D are *equivalent* if and only if there exists a mapping $f : \mathbb{Z} \to \mathbb{Z}$ such that:

- for every integer x, $f(-x) = -f(x)$,
- whenever $f(x) \neq 0$, the integers x and $f(x)$ have the same sign (both positive or both negative),

and for all $1 \leq i \leq 2k$ we have $d_i = f(c_i)$. In other words, if we replace each element c_i of the sequence C by $f(c_i)$, we obtain the sequence D. The values of f on integers that do not occur in the sequences may be chosen arbitrarily.

Input

The first line contains a single integer t ($1 \leq t \leq 10^5$), the number of test cases. For each test case:

The first line contains an integer n ($1 \leq n \leq 5 \cdot 10^5$), denoting that the length of sequence A is $2n$.

The second line contains $2n$ integers, denoting $a_1, a_2, \ldots, a_{2n}$.

The third line contains an integer m ($1 \leq m \leq 5 \cdot 10^5$), denoting that the length of sequence B is $2m$.

The fourth line contains $2m$ integers, denoting $b_1, b_2, \ldots, b_{2m}$.

The sequences A and B meet the requirements in the statement.

The total size of all test cases is bounded by $\sum n + \sum m \leq 10^6$.

Output

For each test case, if it is possible to make A equivalent to B, output "`YES`"; otherwise, output "`NO`".

Examples

standard input	*standard output*
4 1 1 -1 2 -1 -2 2 1 1 1 1 1 -1 1 4 1 2 -1 3 -4 2 3 -4 4 1 2 -1 -2 4 3 4 3 4 4 3 -1 -2 -3 2 1 -4 3 -1 1 3 -3 2 2	YES NO YES NO
5 4 1 2 -3 -2 4 -1 3 -4 6 -3 5 2 3 -2 -5 1 -6 -4 -1 4 6 5 -2 4 -4 3 2 -1 -5 1 -3 5 5 3 -4 1 -5 4 -1 2 -2 -3 5 4 -2 1 4 2 -4 3 -1 -3 5 -1 -5 -2 2 1 -4 -3 3 4 5 3 2 -1 -1 -3 -2 3 4 -4 -2 -1 4 1 -2 3 -3 5 3 5 -5 -2 2 -1 -3 -4 1 4 5 3 5 -2 -5 -1 2 -4 1 4 -3	YES YES NO YES NO

Note

In the first example:

In the first test case, $(1,-1) \xrightarrow{op\ 3} (-2,2,1,-1) \xrightarrow{op\ 1} (-1,-2,2,1)$.

In the third test case, suppose $x=3$, $y=3$, $S_1=(1,2,-1)$, $S_3=(-4,2)$, $S_4=(-4)$, and S_2 is empty. Then $A=(S_1,x,S_2,S_3,y,S_4)$, and $A \xrightarrow{op\ 5} (1,2,-1,-2,4,3,4,3)$.