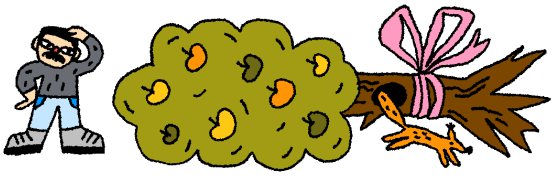
Problem A. Apple Tree

Input file:	*standard input*
Output file:	*standard output*
Time limit:	8 seconds
Memory limit:	256 mebibytes



Aleksei received an apple tree of size n as a birthday present. As a competitive programmer himself, he decided to calculate the number of *k-apple pies* on the tree.

An *apple tree* of size n is defined as a rooted tree with n vertices, rooted at vertex 1, where each vertex has **strictly fewer** direct children than its parent (except the root that doesn't have any parent).

A *k-apple pie* is a pair (v, S) where:

- $v \in V$ is a vertex of the tree (the *center*),
- $S \subset V \setminus \{v\}$ is a set of vertices of size $k - 1$,
- there exists an integer d such that for every $u \in S$, $\text{dist}(v, u) = d$.

The distance $\text{dist}(u, v)$ is defined as the number of edges on the simple path between vertices u and v.

In other words, a k-apple pie consists of one vertex v (the center) and $k - 1$ other vertices that are all at the same distance from v.

Your task is to compute the number of k-apple pies in the given tree.

Since the answer may be large, output it modulo 998 244 353.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

Each test case begins with a line containing two integers n and k ($2 \le n \le 5 \cdot 10^5$, $1 \le k \le n$): the number of vertices in the apple tree and the size of the apple pie.

The next line contains $n - 1$ integers $p_2, p_3, \ldots, p_n$ ($1 \le p_i < i$), where p_i denotes the parent of vertex i in the apple tree.

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

Output

For each test case, output a single integer: the number of k-apple pies modulo 998 244 353.

Example

standard input	*standard output*
3 3 3 1 1 5 2 1 1 2 3 6 3 1 1 1 2 2	1 20 16

Note

In the first test case, the only existing pie is centered at vertex 1, and the corresponding set is $S = \{2, 3\}$.

Problem B. Balatro

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	256 mebibytes

Do we really have to explain the rules of *Balatro*? Just play it yourself. And remember to stop before the next contest.

For those unfamiliar with the rules: you have n cards in your hand. Each card has two associated values: a_i and b_i. You can select any subset S of exactly k cards and play them together to obtain a score calculated as:

$$\left(\sum_{i \in S} a_i\right) \cdot \left(\sum_{i \in S} b_i\right)$$

Your task is to determine the maximum possible score you can achieve in a single play using exactly k cards.

It is additionally known that the deck is *balanced*, meaning no card has both a_i and b_i simultaneously too high. Specifically, for every card, $\min(a_i, b_i) \le 100$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

Each test case begins with a line containing two integers n and k ($1 \le n \le 10^5$, $1 \le k \le \min(n, 5)$): the total number of cards and the number of cards to select for a single play.

Each of the next n lines contains two integers a_i and b_i ($1 \le a_i, b_i \le 10^9$), representing the values of a card. For every card, $\min(a_i, b_i) \le 100$.

The total number of cards across all test cases does not exceed 10^5.

Output

For each test case, print a single integer: the maximum possible score that can be obtained by selecting exactly k cards.

Examples

standard input	*standard output*
1 5 5 1 1 2 2 3 3 4 4 5 5	225
1 6 5 1 1 2 6 3 5 4 4 5 3 6 2	400

Note

In the first test case, we use all the cards. The score is $(1+2+3+4+5) \cdot (1+2+3+4+5) = 15^2 = 225$.

In the second test case, we have to drop one card. The set with cards $[2, 3, 4, 5, 6]$ has the value 400.

Fun fact: In the real Balatro game, a_i of a card is based on it's *poker* value, and b_i can be 0, 4, or 20 (20 can happen with a 20% chance). Also, there are hundreds of bonus cards and rules that influence how the sums of a and b are calculated.

Problem C. Classement Nationale

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	256 mebibytes

Orienteering is a sport where athletes race through forests using a map to visit control points. After arriving in France, Kostya wanted to understand the local rating system called "Classement Nationale" (CN). Here is a moderately simplified model of this rating system.

In our model of the French orienteering system, there are the following elements:

- A total of n athletes, numbered 1 through n. Kostya is athlete number 1.
- A list of m competitions (races) in the order of their occurrence. Competition j occurs on day d_j, and has a flag $s_j \in \{0, 1\}$ indicating "special" status (for example, French Championships).
- A list of k_j participants who took part in competition j. The i-th participant is athlete number a_i who finished the race with time (result) $R_{a_i,j}$. Some athletes may be disqualified (DSQ): they appear in the list with $R_{a_i,j} = 0$. Except for the DSQ case, the less the $R_{a_i,j}$ value is, the better the performance of an athlete.
- Each athlete i has a base rating $CN_{i,\text{base}}$ (their rating before any competition they participated in).
- Each day is attributed to a year, each year being 365 days long. So, days $[0; 365)$ form the first year, days $[365; 730)$ form the second year, and so on.
- During each year, there exists a normalization coefficient K_{norm}. During the first year, $K_{\text{norm}} = 10\,000$.

At each moment of time t, athlete i has a defined rating $CN_{i,t}$. Ratings evolve over time according to the following process.

1. For each competition on day d_j:
   (a) Fetch each participant's rating from 15 days ago: $CN_{i,(d_j-15)}$.
   (b) Calculate $F_j = |\{i : R_{i,j} \neq 0\}|$, the number of participants who finished the race without being disqualified.
   (c) Compute the *RaceScore*:

$$N_{\text{infl}} = \max\big(3, \lfloor 2F_j/3 \rfloor\big),$$

$$\text{and if } F_j \geq N_{\text{infl}}, \text{ then } \mathit{RaceScore}_j = \left\lfloor \frac{1}{N_{\text{infl}}} \sum_{p=1}^{N_{\text{infl}}} \big(R_{Pos_p,j} \cdot CN_{Pos_p,(d_j-15)}\big) \cdot \frac{10\,000}{K_{\text{norm}}} \right\rfloor$$

   where $Pos_1, \ldots, Pos_{F_j}$ lists non-disqualified participants in increasing order of $R_{i,j}$ (ties are broken in a way that an athlete with a higher CN value is earlier in rankings). If $F_j < N_{\text{infl}}$, the race is unrated and no performances are recorded.
   (d) In case the race is rated, each of the k_j athletes who took part gets a record of participation. Athlete a_i has the following performance value:

$$\mathit{Performance}_{a_i,j} = \begin{cases} -1 & \text{if } R_{a_i,j} = 0 \text{ (DSQ)}, \\ \left\lfloor \frac{\mathit{RaceScore}_j}{R_{a_i,j}} \right\rfloor & \text{otherwise.} \end{cases}$$

   This performance (along with competition date d_j and special flag s_j) is added to athlete i's history.

2. At any day t, an athlete i's current $CN_{i,t}$ is defined by their recorded performances. Let H_i be the multiset of all performances in days $(t-365, t]$.

   (a) If $H_i = \varnothing$, then:

   $$CN_{i,t} = \begin{cases} CN_{i,\text{base}} & \text{if athlete } i \text{ has no recorded performances for all times in } (-\infty, t], \\ 0 & \text{otherwise.} \end{cases}$$

   (b) Otherwise:

   - Sort H_i in non-increasing order.
   - Let $h = \min(\max(\lfloor 0.4 \cdot |H_i| \rfloor, 4), |H_i|)$. Then:
     i. Take the h largest **non-negative** scores into multiset S. In case of ties performance in special race ($s_j = 1$) should be taken first.
     ii. Construct a multiset $S' = S$. If $|S| > 1$, drop the very best score from S' unless there exists a special race ($s_j = 1$) with the very best score. If $|S| \le 1$, S' is not changed under any circumstances.
     iii. Compute avg: the average of the scores in S'. In case $|S'| = 0$, let $avg = 0$.
     iv. If $|S| < 4$, there will be a penalty of 2% per missing race (for example, 3 races $\to$ 2%, 2 races $\to$ 4%, 1 race $\to$ 6%). If there were DSQ races in H_i, this penalty should be reduced by 2% regardless of the number of DSQ races. We define the effect of x% penalty on number y as $\frac{100-x}{100} \cdot y$.
   - The floored avg after applying the penalty is $CN_{i,t}$. Please note that this value is an integer, and was floored from the corresponding rational number exactly once in the end.

3. Additionally, whenever $t \equiv 0 \pmod{365}$ for all years but the first, a recalibration is performed (before any of the races on that day, if any):
   - Let $M = \max_i CN_{i,t-1}$.
   - If $M = 0$, $K_{\text{norm}} = 10\,000$.
   - Otherwise, $K_{\text{norm}} = M$.

Orienteering is not the most popular sport, so athletes want as many people coming to the competitions as possible. Kostya has a hunch that this complicated system has the opposite effect.

So Kostya wants to show the French Orienteering Federation that by skipping some races he actually attended, he could end up with different ratings after the last competition of the season. In other words, he is interested in different possible values of CN_{1,d_m} after all updates have been processed if he was not listed in the results of some races. As orienteering is an individual sport, all $R_{i,j}$ do not influence each other.

Given the full data of n athletes, m competitions, and Kostya's personal attendance list, compute every distinct final rating CN_{1,d_m} achievable by choosing a subset of the races Kostya attended (including an empty subset). Output the ratings in strictly decreasing order.

Input

The first line contains two integers n and m ($1 \le n, m \le 1000$): the number of athletes and the number of competitions.

The second line contains n integers $CN_{i,\text{base}}$ ($1 \le CN_{i,\text{base}} \le 12\,000$): base ratings for the athletes.

The third line contains m integers $d_1, \ldots, d_m$ ($1 \le d_i \le 2000$, $d_i \le d_{i+1}$): competition days for the races.

The fourth line contains m integers $s_1, \ldots, s_m$ ($0 \le s_i \le 1$): special flags for the races.

Then follow n blocks, one per athlete $i = 1, \ldots, n$:

- A line with an integer n_i ($0 \le n_i \le 10$), the number of races that athlete i participated in.

- Each of the next n_i lines contains two integers b_j and R_{i,b_j} ($1 \le b_j \le m$, $600 \le R_{i,b_j} \le 9000$ or $R_{i,b_j} = 0$): competition number and result in seconds. For each player, all b_j are distinct.

Output

On the first line, print a single integer c: the number of distinct achievable ratings. On the next c lines, print the achievable values of CN_{1,d_m} in decreasing order, one per line.

Examples

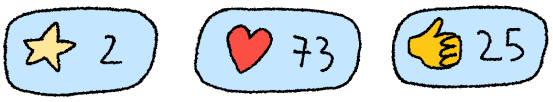
standard input	*standard output*
3 4 6200 7800 10200 90 100 300 464 1 1 1 1 4 1 1950 2 1940 3 900 4 1800 4 1 1920 2 1800 3 844 4 1800 4 1 1860 2 1670 3 769 4 1800	15 9156 8120 8016 7981 7920 7828 7719 7433 6983 6965 6960 6899 6600 6200 0
6 2 10118 10485 10427 9158 9307 10044 356 365 0 0 2 1 1105 2 791 1 2 960 1 1 929 1 1 841 0 2 1 1043 2 1195	4 11360 10118 8068 7900

Note

We will explain two of the achievable CN values for the first example. Skipping all the 4 races would result in $CN_{1,d_m} = CN_{1,\text{base}} = 6200$ for Kostya. If Kostya attends only the first race on day 90, by the day 464 it would be gone from the performances being considered, thus his $CN_{1,d_m} = 0$. Other possible results are given in the output.

Problem D. Digit Division

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes



You are an infamous blogger with a wide network of channels and groups in some troubled messenger founded by the French entrepreneur Du Rove. To have a recognizable author's style, you give all channels some number-related names, including `itstwofortyfive`, `four`, `two seven three two five`, `twenty five`, `0x6ABD`, and `signed integer overflow`.

You want to launch a new channel, but you ran out of numbers, so you need a new one.

Today your favourite number is k, you want to choose some number n as a new channel name.

You will be satisfied with a channel name n if n has the following properties:

- the number n is divisible by k,
- the sum of all digits in n is k.

You already found all channel names for small values of k, so now you need a program to work with larger numbers.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). The description of the test cases follows.

Each test case is given on a single line containing one integer k ($1 \le k \le 10^5$).

The sum of k over all test cases does not exceed $2 \cdot 10^5$.

Output

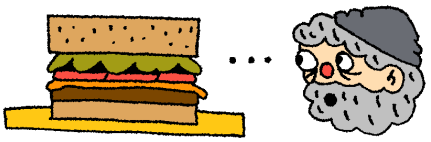
For each test case, output n ($1 \le n \le 10^{20k}$). In case there are multiple answers, output any one of them.

Example

standard input	*standard output*
4 1 9 12 90	1 81 12121212 27325000144500025000400021474836470000

Problem E. Euclid in Manhattan

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



Due to an unfortunate time-travel accident, Euclid finds himself wandering the streets of modern-day Manhattan. To his dismay, he discovers that people are measuring distances using the so-called *Manhattan metric*!

"This isn't the shortest path at all, and I can prove that!", he shouted. Passers-by didn't care about the crazy man.

Manhattan is structured as a grid formed by the Cartesian product of two axes, A and B. The positive direction along axis A goes **right**, and along axis B goes **down**.

Each axis is partitioned into segments that alternate between:

- Residential zones: these are solid blocks that cannot be traversed,
- Streets (or avenues): gaps between the residential zones, where movement is allowed.

The segments are numbered starting from 1, and the pattern alternates: segments with odd indices are residential, and even indices are gaps. The number of segments is always odd.

Additionally, no street or avenue is wider than any dimension of any residential zone. Formally, among the segments on both axes, the smallest segment with an odd index is no less than the largest segment with an even index.

We define a point (x, y) to be *inside a building* if it lies within a horizontal residential segment (from axis A) **and** within a vertical residential segment (from axis B). These building areas are impassable. You may move freely elsewhere, as long as your path does not pass through the interior of any building. Touching the edges or corners is allowed.

Euclid wants to get from the top-left corner of the grid to the bottom-right corner, moving through open areas only, and using Euclidean distance. Help him find the shortest possible path!

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 100$). The description of the test cases follows.

Each test case is described as follows:

- A line containing two integers n and m ($1 \le n, m \le 2000$): the number of residential segments along the horizontal and vertical axes, respectively.
- A line with n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$): the widths of the horizontal residential segments.
- A line with $n-1$ integers $b_1, b_2, \ldots, b_{n-1}$ ($1 \le b_i \le 10^6$): the widths of the horizontal gaps (streets) between residential blocks. For input consistency, we add an extra zero at the end of this line. It should be read and ignored.
- A line with m integers $c_1, c_2, \ldots, c_m$ ($1 \le c_i \le 10^6$): the heights of the vertical residential segments.
- A line with $m-1$ integers $d_1, d_2, \ldots, d_{m-1}$ ($1 \le d_i \le 10^6$): the heights of the vertical gaps (avenues) between residential blocks. For input consistency, we add an extra zero at the end of this line. It should be read and ignored.

In each test case, $\max(b_1, \ldots, b_{n-1}, d_1, \ldots, d_{m-1}) \le \min(a_1, \ldots, a_n, c_1, \ldots, c_m)$.

The sum of $n \cdot m$ over all test cases does not exceed $4 \cdot 10^6$.

Output

For each test case print a single real number: the minimum Euclidean distance needed to travel from the top-left corner to the bottom-right corner, without passing through the interior of any building.

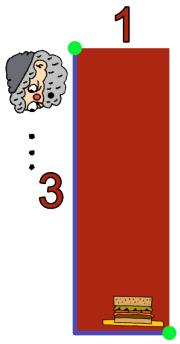
Your answer must have an absolute or relative error of at most 10^{-9}.

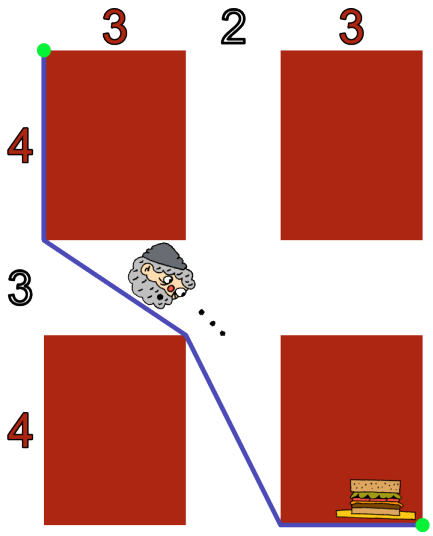
Example

standard input	*standard output*
2 1 1 1 0 3 0 2 2 3 3 2 0 4 4 3 0	4.0000000000 15.7147766421

Note

You can see the illustrations for the two example test cases below.





Problem F. Framboise 2

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

Your summer house plot is a rectangular area surrounded by a fence along its perimeter. The plot is divided into 2 horizontal and n vertical rows of square zones of size 1×1.

In the centers of some k square zones grow raspberry plants, each having four stems. Other square zones are empty. You want to tie the plants to the fence (being a gardener, you have your own reasons to do so).

To tie a plant, you can lay its stem in a straight line along the ground toward the fence and tie it to the corresponding segment of the fence. You may only stretch stems in directions parallel to the fence (up, down, left, or right). Such stems are called *active*.

A *tying configuration* is a set of active stems that connect the plants with the fence. Each plant can be connected more than once, or not be connected at all.

We call a tying configuration *unstable* if a stem lies on top of a plant (except the plant it originated from) or if a stem has a common point with any other stem (except when they both originate from the same plant). All other configurations are *stable*.

Several years ago, you found the optimal configuration that tied the maximum possible number of plants. But today, you asked yourself a different question: what is the total number of stable configurations? As this number might be large, find it modulo 998 244 353.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

Each test case starts with a line containing two integers n and k ($1 \le n \le 10^9$, $1 \le k \le \min(10^6, 2 \cdot n)$): the size of the plot and the number of raspberry plants.

Each of the next k lines contains two integers r_i and c_i ($1 \le r_i \le 2$, $1 \le c_i \le n$): the row and column of the square zone containing a raspberry plant. Each pair (r_i, c_i) appears at most once per test case.

The sum of k over all test cases does not exceed 10^6.

Output

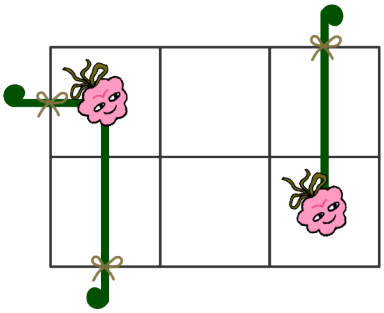
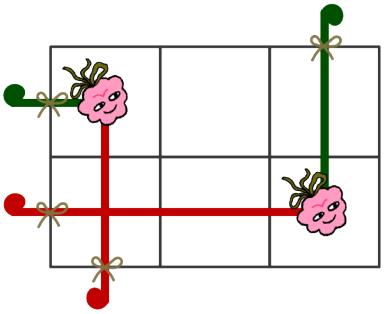
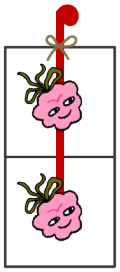
For each test case, print a single integer: the total number of stable configurations. As this number might be large, output it modulo 998 244 353.

Example

standard input	*standard output*
2 3 2 1 1 2 3 1 2 1 1 2 1	144 64

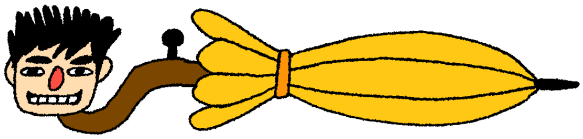
Note

We provide three illustrations for the example:

Stable configuration for the first test case	
Unstable configuration for the first test case. Two stems intersect	
Unstable configuration for the second test case. Stem from the bottom raspberry plant crosses the other raspberry plant	

Problem G. Goofy Songs

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes



KiKoS is a fan of the TV series "How I Met Your Mother". Recently, he discovered a Spotify playlist with its soundtrack.

Alongside the undeniable hits "Let's Go to the Mall" and "Nothing Suits Me Like a Suit", he found a very goofy song: "Bang bang bangity bang". This is basically the only phrase in the entire song. To make it longer, you just say the same line, but prepend "I said" at the beginning, and then you can sing the whole thing again.

As a newbie songwriter, KiKoS wants to reuse this structure for a new song. Formally, given a non-empty word $\mathbb{S}$ consisting of lowercase English letters, the $\mathbb{S}$-song is composed of a **block of two lines** that can be repeated multiple times using the same word $\mathbb{S}$. Each block looks like:

$\mathbb{S}$, $\mathbb{S}$ $\mathbb{S}$ity $\mathbb{S}$
i said $\mathbb{S}$, $\mathbb{S}$ $\mathbb{S}$ity $\mathbb{S}$

Given a large multiline text, determine the maximum possible amount of characters in some sequence of consecutive blocks that forms a valid song.

Input

The input consists of multiple lines, consisting of lowercase English letters, commas, and spaces. Each line ends with a newline character. The total number of characters including endlines does not exceed 10^5. There are no empty lines, and lines do not start or end with a space.

Output

Print a single integer: the maximum possible amount of characters in some consecutive sequence of blocks that matches the song pattern. This includes newline characters at the end of each line of the sequence, including the last line of the song. Each newline is considered to be 1 character: pay attention to this if you have a Windows setup.

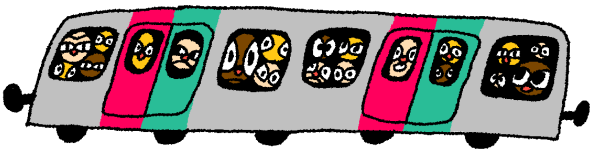
If there are no blocks matching the song pattern, output -1.

Examples

standard input	*standard output*
bang, bang bangity bang i said bang, bang bangity bang	55
bang, bang bangity bang i said bang, bang bangity bang bank, bank bankity bank i said bank, bank bankity bank bank, bank bankity bank i said bank, bank bankity bank	110
grandmasters are red, experts are blue, parsing problems are bad, but we believe in you	-1

Problem H. Heure de Rush

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes



You are managing a metro train of n cars. There are $n \cdot k$ people unevenly distributed over the cars; more precisely, there are h_i people in car i. You want to balance the load and have exactly k people in every car.

At every stop, some people (probably none) can leave their car and run to another one, but they cannot move more than d cars in one run.

It's up to you to decide who runs where and when. How many stops do you need to achieve your goal?

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains two integers n and d ($1 \le d \le n \le 10^6$): the number of cars and the distance one person can run at once.

The second line of each test case contains n integers $h_1, h_2, \ldots, h_n$ ($0 \le h_i \le 10^9$): the distribution of people over the train. The sum of h_i is divisible by n.

The sum of n over all test cases does not exceed 10^6.

Output

For each test case, output a line with a single integer: how many stops you need.

Example

standard input	*standard output*
3 1 1 5 3 2 3 0 0 10 1 0 0 10 0 0 0 0 10 0 0	0 1 2

Note

In the first test case, there is a single car, so people are evenly distributed from the start.

In the second test case, before the start, there are 3 people in the first car: Aleksei, Dima, and Kostya. On the first stop, Dima runs to the second car, and Kostya runs to the third one, so one stop is enough to make every car have one person.

Problem I. Infrared

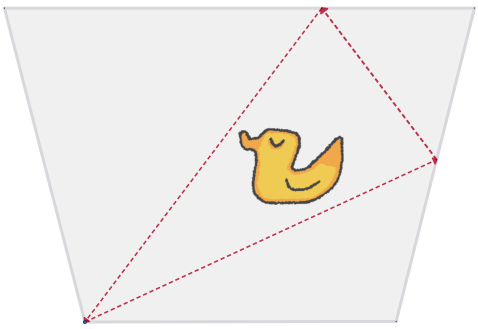
Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	256 mebibytes

You've just added the crown jewel to your rubber duck collection: a royal rubber duck. Sadly, Akulyat has already caught the scent and is speeding toward your vault to steal it.

To protect it, you've placed the duck inside the safest room of your vault. The room is shaped like a convex polygon with n vertices, and thus has n walls. There is a secure door located at vertex 1. This door is equipped with an infrared laser system that protects walls 1 (between vertices 1 and 2) and n (between vertices n and 1).

However, you're worried about the unprotected walls 2 through $n-1$. Fortunately, the security system also has a powerful laser emitter and receiver installed at vertex 1. You want to aim the laser in such a way that:

- it starts at vertex 1,
- reflects in this exact order off wall 2, then wall 3, and so on until it reflects off wall $n-1$,
- and finally returns precisely to vertex 1.



All reflections follow the laws of ideal optics (angle of incidence equals angle of reflection), and the polygon is convex. You need to determine if it is possible to align the laser in the described way. If so, compute the required firing angle.

The base direction (angle 0°) is the positive x-axis, and angles are measured in degrees counter-clockwise.

Important: To not mess up the optics, you decided to have all reflection points strictly inside their respective walls: more precisely, to be at least 10^{-5} **relative** units away from both endpoints. Formally, the valid points of reflection p for the wall $[a, b]$ are of the form $p = \alpha a + (1-\alpha)b$, where $\min(\alpha, 1-\alpha) \geq 10^{-5}$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 1000$). The description of the test cases follows.

For each test case, the first line contains a single integer n ($3 \leq n \leq 20$): the number of vertices of the convex polygon.

Each of the next n lines contains two integers x_i and y_i ($-10^9 \leq x_i, y_i \leq 10^9$): the coordinates of the vertices. The vertices are given in counter-clockwise order and form a strictly convex polygon.

Output

For each test case, output a single line "NO" if there is no answer.

Otherwise output two lines. The first line must contain "YES". On the second line, print a real number d ($-180 \le d \le 180$): the angle (in degrees) relative to the positive x-axis (counter-clockwise), at which the laser should be fired towards wall 2.

The trajectory of the ray should hit all the walls in required order at points inside the walls, and end in the 10^{-7}-neighborhood of vertex 1. The 10^{-7}-neighborhood of vertex 1 is defined as any point (x, y) on the walls 1 or n that follows $(x_1 - x)^2 + (y_1 - y)^2 < (10^{-7} \cdot L)^2$, where L is the total length of polygon sides.

If multiple angles satisfy the condition, output any one suitable angle.

Examples

standard input	*standard output*
2 3 1 2 0 0 2 0 4 0 0 1 0 1 1 0 1	YES -90 NO
3 4 0 0 4 0 5 4 -1 4 5 -809016994 -587785252 309016994 -951056516 1000000000 0 309016994 951056516 -809016994 587785252 6 -1000000000 0 -500000000 -866025404 500000000 -866025404 1000000000 0 500000000 866025404 -500000000 866025404	YES 24.623564786163612980174342226292 YES 9.732301397707778775440778940009 NO

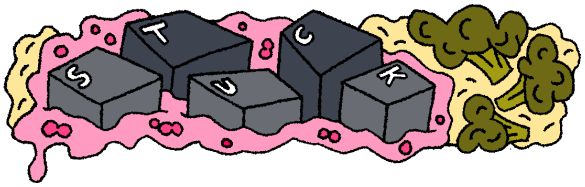
Note

Illustration for the first test case of the second example is provided in the statement.

It is guaranteed that, for each test with answer "YES", there exists a shooting angle that has all its reflections at points $p = \alpha a + (1 - \alpha)b$, where $\min(\alpha, 1 - \alpha) \ge 10^{-5} + 10^{-9}$). It is also guaranteed that, for each test with answer "NO", there are no shooting angles that are *almost* valid: this means that at some point there is either no intersection, or $\min(\alpha, 1 - \alpha) < 10^{-5} - 10^{-9}$.

Problem J. JamBrains

Input file:	*standard input*
Output file:	*standard output*
Time limit:	3 seconds
Memory limit:	256 mebibytes



It's not enough to write the code, you still have to fix three hundred bugs

Put your cursor at the end of a line in the middle of your code. Press `RUL` *arrows. Now repeat it many times. You learned a convenient navigation trick.*

Have you eaten meatballs with cranberry jam? Kostya adores them, especially while solving problems at midnight. In fact, he ate too much in front of his computer: some cranberry jam dripped out of the jar and ruined his keyboard. But he still has to finish solving the problem. And, unlike his keyboard, he doesn't want to be stuck!

The code file that Kostya used consists of n lines numbered from 1 to n, where line i contains s_i possible cursor positions from $(i, 1)$ to (i, s_i), denoting the j-th position on the i-th line as (i, j).

Code navigation with some keys stuck is a bit tricky. Fortunately, Kostya still has Up and Right arrows working:

- Up ($\uparrow$): if the cursor is at position (i, j), then if $i = 1$, the cursor stays in place; otherwise, it moves to position $(i - 1, \min(j, s_{i-1}))$, where s_{i-1} is the length of line $i - 1$.
- Right ($\rightarrow$): if the cursor is at position (i, j), then if $j < s_i$, the cursor moves to $(i, j + 1)$; otherwise, it moves to $(i + 1, 1)$; except when at (n, s_n), then it stays in place.

Kostya has a special technique. He picks an initial cursor position and starts to perform an infinite sequence of key presses: u times Up, followed by r times Right, then again u times Up, followed by r times Right, and so on.

For example, let $n = 5$, $u = 2$, $r = 5$, and $s = \{5, 2, 4, 3, 1\}$. Starting from position $(4, 3)$, the first few key presses will move the cursor as follows:

$$(4, 3) \uparrow (3, 3) \uparrow (2, 2) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (4, 1) \ldots$$

A position is called *successful* if, starting from this position and performing his infinite sequence, Kostya will sooner or later have the cursor on the first line.

Your task is to help Kostya count how many successful starting positions exist in his code.

His code isn't finished though, so you also need to handle q updates. In each update, you get two values, pos and x, and you have to assign s_{pos} a new value x. You need to give the answer to the problem before any updates and after processing each update.

Input

The first line contains three integers: n, u, and r ($1 \le n \le 10^5$, $1 \le u, r \le 10^{18}$).

The second line contains n integers $s_1, s_2, \ldots, s_n$ ($1 \le s_i \le 10^{12}$): the number of positions in each line.

The third line contains a single integer q ($1 \le q \le 10^5$), the number of updates.

Each of the next q lines contains two integers, pos and x ($1 \le pos \le n$, $1 \le x \le 10^{12}$): the updates.

The updates are consecutive, so each update is applied to the current state of the code, not the initial one.

Output

Output $q + 1$ integers, each on a separate line: the number of successful starting positions before all the updates and after each one of them.

Examples

standard input	*standard output*
5 2 5 5 2 4 3 1 1 1 1	15 11
5 2 10 5 2 4 3 1 1 2 3	11 12
10 1 42 169 42 42 42 42 42 42 42 42 42 1 2 43	211 254

Note

In the first example, before any updates, starting from position $(5, 1)$ and pressing "2 times Up followed by 5 times Right" in a loop:

Round 1: $(5, 1) \uparrow (4, 1) \uparrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (4, 1) \rightarrow (4, 2)$

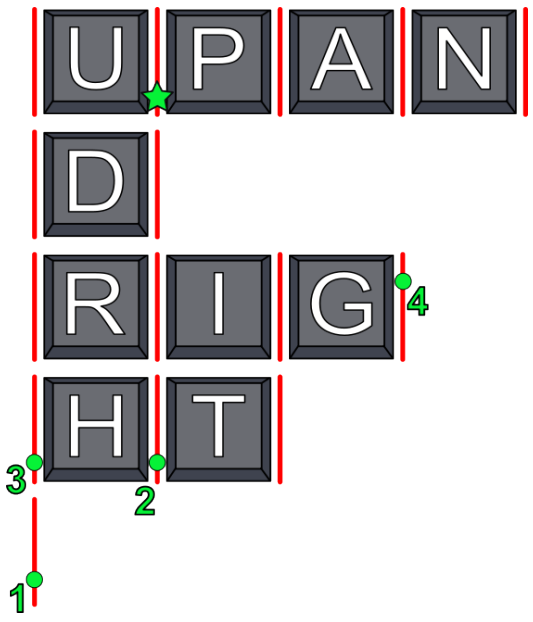
Round 2: $(4, 2) \uparrow (3, 2) \uparrow (2, 2) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4) \rightarrow (4, 1)$

Round 3: $(4, 1) \uparrow (3, 1) \uparrow (2, 1) \rightarrow (2, 2) \rightarrow (3, 1) \rightarrow (3, 2) \rightarrow (3, 3) \rightarrow (3, 4)$

Round 4: $(3, 4) \uparrow (2, 2) \uparrow (\mathbf{1}, \mathbf{2}) \rightarrow \ldots$

Thus, $(5, 1)$ is a successful position.

On the image below you can see the starting positions of each round marked with the green dots and the final position marked with the green star.



Problem K. k Operations

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

Usually, whenever Akulyat encounters a problem involving queries and mysterious operations, he immediately delegates it to KiKoS. To educate Akulyat during the Pafos camp, KiKoS decided to give him this problem. But, well, the plan failed: Akulyat found the sea and was spending all his time there. Well, at least now there is an unsolved problem for you.

You are given an array $a = [a_1, a_2, \ldots, a_n]$ of n positive integers.

We define $f_k(b_1, b_2, \ldots, b_m)$ to be the minimum possible product of the array $[b_1, \ldots, b_m]$ that can be obtained by applying the following operation at most k times:

- Select any index i such that $b_i > 1$ and assign $b_i := b_i - 1$.

Your task is to process q queries. Each query is given as three integers ℓ, r, and k, and asks you to compute the value of $f_k(a_\ell, a_{\ell+1}, \ldots, a_r)$: the minimum possible product of the subarray a_ℓ through a_r after applying at most k operations, as described above.

Since the answer may be large, output it modulo 998 244 353.

Input

The first line contains two integers n and q ($1 \le n \le 2 \cdot 10^5$, $1 \le q \le 5 \cdot 10^5$): the size of the array and the number of queries.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 998\,244\,352$).

Each of the next q lines contains three integers ℓ, r, and k ($1 \le \ell \le r \le n$, $0 \le k \le 10^{18}$) describing a query on the subarray from index ℓ to r, with at most k allowed operations.

Output

For each query, print one integer: the value of $f_k(a_\ell, \ldots, a_r)$ modulo 998 244 353.

Example

standard input	*standard output*
5 2 1 2 3 4 5 1 2 3 4 5 1	1 15

Note

The product is computed after all operations are applied, and only the final result should be taken modulo 998 244 353.

Problem L. Lice Hopping

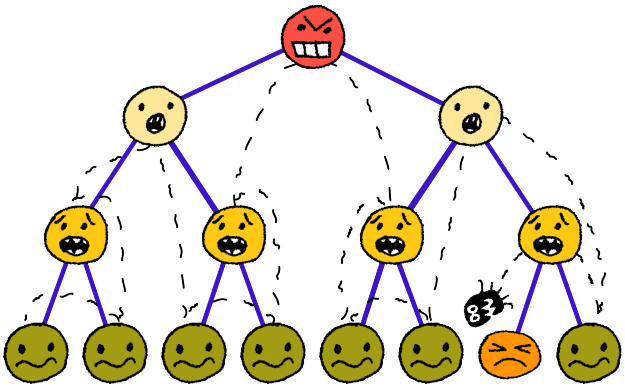
Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

Today you are a lice jumping from head to head to find the best hair for you to live.

The group of people sitting in the room you want to explore can be represented as a tree (connected graph without cycles) of size n, where each head is a vertex, and the edge exists if the two heads are at a jumping distance 1 from one another.

You want to visit the heads of every person in the room. To not make anyone suspicious, you decided to visit each head only once. Before starting your journey, you will train your skill of lice hopping, which will allow you to make longer jumps. If you spent d days on mastering hopping, you are now able to jump a distance no more than d in the tree. Here, the distance $\text{dist}(u, v)$ is defined as the number of edges on the simple path between vertices u and v. A head is considered visited either if it was a starting head or if it was already jumped on. Please note that heads jumped over (between the start and end head of a single jump) are not marked as visited.

What is the minimum number of training days you need to train to be able to visit all the heads? You can choose any starting head you want.



Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 2 \cdot 10^5$). The description of the test cases follows.

The first line of each test case contains one integer n ($2 \le n \le 10^6$), the size of the tree.

Each of the next $n - 1$ lines contains two integers u and v ($1 \le u, v \le n$, $u \ne v$): the vertices connected by an edge of the tree.

The sum of n over all test cases doesn't exceed 10^6.

Output

For each test case, output a line with a single integer: the minimum number of training days needed to be able to jump over the given tree.

Example

standard input	*standard output*
4 3 1 2 1 3 5 1 2 1 3 1 4 1 5 7 1 2 1 3 2 4 2 5 3 6 3 7 15 1 2 1 3 2 4 2 5 3 6 3 7 4 8 4 9 5 10 5 11 6 12 6 13 7 14 7 15	1 2 2 2

Note

For the first test case, you can perform the following sequence of hops: $3 \to 1 \to 2$. For each jump, the distance is 1.

The fourth test case describes the graph from the statement. With $d = 2$, a valid sequence of hops is: $8 \to 9 \to 4 \to 2 \to 10 \to 11 \to 5 \to 1 \to 6 \to 12 \to 13 \to 3 \to 15 \to 7 \to 14$. The distances are: $2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 1, 1$. So 2 training days are enough.

Problem M. Manhattan Graph

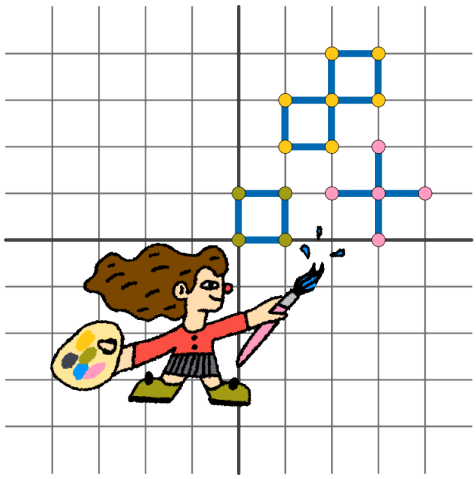
Input file:	*standard input*
Output file:	*standard output*
Time limit:	10 seconds
Memory limit:	1024 mebibytes

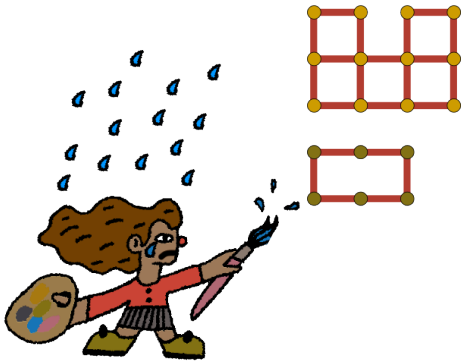
Lucy is preparing the illustrations for Xeppelin Contest. For one of the problems, she was asked to draw a graph. The issue is, it is not easy to draw a nice graph.

Lucy thinks that the best place to draw an undirected connected graph with n vertices and m edges is checkered paper. Each vertex of the graph will be an integer point of the grid (x, y), and each edge will be a line connecting either points $(x, y) \leftrightarrow (x + 1, y)$ or $(x, y) \leftrightarrow (x, y + 1)$ for some integers x and y. She can just highlight some parts of the checkered pattern, and the illustration will be ready!

However, it would be nice if the drawing preserved the distances in the graph. Formally, consider points with integer coordinates on a two-dimensional plane. A *drawing* is an array of n such points $p_1, \ldots, p_n$. A drawing is *nice* if $\text{dist}(i, j) = \text{Manhattan}(p_i, p_j)$ for all possible i and j. Here, $\text{dist}(u, v)$ is defined as the number of edges on the shortest path between vertices u and v in the graph, while $\text{Manhattan}(p_i, p_j)$ is defined as the Manhattan distance between two integer points p_i and p_j on the plane.

Find a nice drawing for the given graph or indicate that it does not exist.





Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 50\,000$). The description of the test cases follows.

The first line of each test case contains two integers n and m ($1 \le n \le 500\,000$, $n - 1 \le m \le 500\,000$): the number of vertices and edges in the graph.

Each of the next m lines contains two integers u and v ($1 \le u, v \le n$, $u \ne v$): the two vertices connected by an edge of the graph. The graph is connected.

Both the sum of n and the sum of m over all test cases don't exceed $500\,000$.

For the convenience of visual perception, there is an empty line before every test case. It can be ignored.

Output

For each test case, print "`NO`" on a single line if no nice drawing exists.

Otherwise, print "`YES`" on the first line. After that, print an example of a nice drawing. Each of the following n lines should contain two integers x_i and y_i: the coordinates of the i-th point ($-998\,244\,353 \le x_i, y_i \le 998\,244\,353$). If there are several possible answers, print any one of them.

Examples

standard input	*standard output*
3 4 4 1 2 2 3 3 4 4 1 5 4 1 2 1 3 1 4 1 5 7 8 1 2 1 3 2 4 3 4 4 5 4 6 5 7 6 7	YES 0 0 1 0 1 1 0 1 YES 3 1 2 1 3 0 4 1 3 2 YES 1 2 2 2 1 3 2 3 3 3 2 4 3 4
2 6 6 1 2 2 3 3 4 4 5 5 6 6 1 12 16 1 2 1 3 2 4 3 4 3 5 4 6 5 6 6 7 4 8 7 8 7 9 8 10 9 10 8 11 10 12 11 12	NO NO