# Toss the BigOs (SAP 3) Daniel loves board games. He likes to spend some time of his break at work by creating strategies for the trending board game. His most recent addiction is an RPG game called **Nlogonions and Logions**, created by Studios Abertos de Programação (SAP). In this game, instead of rolling a D10 die to calculate the success percentage of an action, the player rolls two dice in a tetrahedron format, called **BigOs**, to calculate the error percentage. The shorter the distance between the BigOs, the lower the error percentage.

To allow BigOs to "stop" in any position, SAP has developed a special table that allows BigOs to stop when they reach the table space, called **BigOs Table**, allowing "to stop" them in the air. The D12 die, used to determine the damage, is not used in this version of the game, since the value is fixed for any action. Daniel wants to know, for a given instant, what is the shortest distance between the two BigOs in order to reduce the error percentage?

**Input**

The input consists of several test cases. The first line of the every input contains an integer **C**, indicating the number of test cases. Each test case consists of eight lines, each describing the coordinate of a vertex of a BigO. The first four lines indicate the vertices of the first BigO and the next four lines indicate the vertices of the second BigO.

Each coordinate description is a line containing three integers **X**, **Y** and **Z** (−103 ≤ **X**,**Y**,**Z**, ≤ 103) indicating the coordinate of a vertex in the BigOs Table. Four vertices always define a BigO of non-zero volume and the two BigOs are always disjoint.

**Output**

For each test case of the input, your program must print a single number, indicating the distance between the two BigOs. The distance between the two BigOs is always greater than zero.

Your answer will be considered correct if the absolute or relative value of the difference between the returned and the expected answer does not exceed **10-6**.

| **Input Sample** | **Output Sample** |
| --- | --- |
| 2 4 -4 -2 -2 -4 -4 1 4 -3 1 0 -4 -2 4 -1 4 4 1 1 -4 0 1 0 -1 -8 -3 0 6 6 -7 -1 5 -10 -4 3 -4 3 4 -4 -4 9 -9 8 2 10 2 9 7 | 1.897367  0.707107 |

# Desk Updates

(SAP 4)

You work at an office with **N** employees and **N** desks. All employees and all desks have an id number from 1 to **N**, and initially each employee is sitting at the desk with the same id as theirs. In other words, employee 1 is sitting at desk 1, employee 2 at desk 2, and so on.

To increase the company's overall productivity, the HR is trying out a variation of Feng Shui at the office, making employees swap desks with each other.

However, the boss sometimes needs to talk with these employees, and he finds it very difficult to keep track of these desk swaps.

It may happen, then, that the boss wants to talk to employee A, so he walks towards desk A, but only to find out that employee C is sitting there. He then assumes that A may be sitting at desk C, so he walks towards desk C. We call this a "redirection", and this can happen several times until the boss finally finds employee A.

In summary, there are two types of events:

* update **A** **B**: the employees **A** and **B** swap desks;
* query **A**: the boss wants to talk to employee **A**.

You got a taks from HR about assessing how this Feng Shui variation process is helping the company. You have to process **Q** events. Whenever the event is of type query, you must figure out how many times the boss was redirected until finding employee **A**. It's guaranteed that he can always find him.

## Input

First line of input will have an integer **N** (2 ≤ **N** ≤ 103), indicating how many employees and desks there are in the office.

The next line will have an integer **Q** (1 ≤ **Q** ≤ 5×103), indicating how many events should be processed.

Following there will be **Q** lines, each representing an event. Each line will have two or three integers. The first integer is **E** (1 ≤ **E** ≤ 2).

If **E** is equal to 1, the event is of type update, and there will be two more integers **A** and **B** (1 ≤ **A**, **B** ≤ **N**) where **A** ≠ **B**.

If **E** is equal to 2, the event is of type query, and there will be one more integer **A**.

## Output

For each event of type query you must print one line, containing one integer, representing how many times the boss was redirected until finding employee **A**.

| **Input Samples** | **Output Samples** |
| --- | --- |
| 3 3 2 1 1 1 2 2 1 | 0 1 |

| 3 3 1 1 2 1 3 1 2 1 | 2 |
| --- | --- |

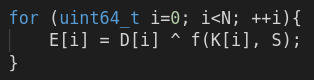
# Farcos and The Xorshift

By Francisco Elio Parente Arcos Filho, UEA BR Brazil

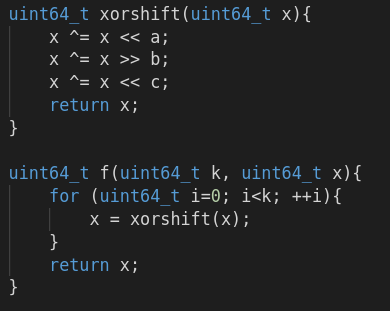
**Timelimit: 2 (SAP 5)**

Xorshift random number generators are a class of pseudorandom number generators that were discovered by George Marsaglia. They have a particularly efficient implementation without using excessively sparse polynomials. They generate the next number in their sequence by repeatedly taking the exclusive or of a number with a bit-shifted version of itself. This makes them extremely fast on modern computer architectures.

In an attempt to create its own encryption, Farcos converts a message into an array of 64-bit integers and applies exclusive or to each element **Ei** with **Ki**-th element of an Xorshift sequence that uses **S** as the seed. In other words, a **E** array was created such that:



An this:



The code snippets shown are implemented in c++.

Provided arrays **E**, **K**, and seed **S**, your task is to decipher the original array **D**.

## Input

The first line of input contains five integers: **N** (0 < **N** ≤ 102), the number of elements in the arrays; **S** (0 < **S** < 264), the seed to Xorshift; **A**, **B** and **C** (0 < **A**, **B**, **C** < 64), the Xorshift parameters shown in the code.

The second line of input has **N** integers **Ei** (0 < **Ei** < 264) separated by a blank space.

The third line of input has **N** integers **Ki** (0 < **Ki** < 264) separated by a blank space.

## Output

The output consists of a single line containing **N** integers **Di** separated by a blank space and representing the deciphered array.

| **Input Samples** | **Output Samples** |
| --- | --- |
| 3 49 63 62 63 48 49 50 98 57 22 | 1 2 3 |

| 4 1 1 2 3 31 80255 345 7119 1 4 2 3 | 4 31396 79 1947 |
| --- | --- |

# Make Jim Win

By Giovanna Kobus Conrado, University of São Paulo BR Brazil

**Timelimit: 3 (SAP 6)**

Jim and Pam are playing a game. They have a collections of rooted trees in which some nodes have some number of stones placed on them. They alternate playing; in each turn the player whose turn it is can take any numbers of stones from a node **V** and put them in **V**'s parent. Whoever can't make a valid move on their turn loses.

Jim knows Pam will want to play first, so he decided to remove some of the edges of the graph beforehand in such a way that he is guaranteed to win if he plays optimally. The more edges Jim removes, the more likely Pam is to be suspicious.

So Jim wants to know what is the smallest number of edges he needs to remove to guarantee his victory assuming Pam starts playing.

## Input

The first line of input consists of one integer **N**, the number of vertices in the graph (1≤**N**≤2×103). The vertices are numbered from 1 to **N**.

The second line contains **N** numbers, the **i**-th of them indicates that there are initially **Si** stones in the node **i** (0≤ **Si** ≤102).

The third line contains **N** numbers, the **i**-th of them indicates the parent node of node **i**. If **i** has no parent this number will be -1.

## Output

The output must consist of a single integer representing the minimum numbers of edges that have to be removed to guarantee Jim'**S** victory, or -1 if it is not possible for him to win.

| **Input Sample** | **Output Sample** |
| --- | --- |
| 4 1 2 3 4 3 3 -1 3 | 3 |

# Folding Paper

By Giovanna Kobus Conrado, University of São Paulo BR Brazil

**Timelimit: 1 (SAP 7)**

You have a huge paper you need to fold. It is **N** centimeters long, **M** centimeters wide and 1 millimeter tall. You can fold it perfectly along both axis. For any integers **X** and **Y** greater or equal to zero, if you fold it **X** times vertically and **Y** times horizontally, it will now be **N/(X+1)**centimeters long, **M/(Y+1)** centimeters wide and **(X+1)×(Y+1)** milimeters tall.

You want to store your paper in a box. For any integer **G**, you can buy a box that can fit any paper no more than than **G** centimeters wide, no more than **G** centimeters long and no more than **K** milimeters tall, for **G** dollars. **K** is a number set by the store that sells the boxes and is the same for all the boxes that they sell.

Now you want to know what is the least you can spend on a box to store your paper.

## Input

The input consists of three, integers, **N**, **M** (1≤**N**,**M**≤1018) and **K** (1≤**K**≤105), as described above.

## Output

The output must consist of a single integer that represent the price in dollars of the cheapest box your paper can fit in if you fold it optimally.

| **Input Sample** | **Output Sample** |
| --- | --- |
| 600 400 7 | 200 |

# Ambiguous Decoding

By Cristhian Bonilha, UTFPR BR Brazil

**Timelimit: 1 (SAP 8)**

Data compression is the process of encoding information using fewer bits than the original representation. It's important, though, that after encoding a message you can also decode it back to its original form.

A code table **T** contains **N** elements instructing how to encode and decode a message. Each element of **T** has two properties **Di** and **Ei**, where **Di** represents a character in its original form and **Ei** represents the same character in its compressed binary form.

A code table is ambiguous for a message if, and only if, after encoding a message **M** to its compressed form R, there is more than one way to decode R into valid messages. In other words, a code table is ambiguous if it's possible to decode R into a message K which is different than **M**.

Given a code table **T** and a message **M**, find out if the code table is ambiguous for that message.

## Input

The first line of input will have an integer **N** (1 ≤ **N** ≤ 27), indicating how many items there are on the code table **T**.

Following there will be **N** lines. Each line will have a string in binary format **Ei** (1 ≤ |**Ei**| ≤ 20) and a character **Di**, meaning that **Ei** is the binary compressed form of the caracter **Di**, and vice-versa.

The last line of input will have a string **M** (1 ≤ |**M**| ≤ 103), representing the message to be encoded.

* The characters **Di** and every character of **M** are either a lowercase latin letter or an underscore '\_'.
* It's garanteed that it's possible to encode **M** using the code table **T**.

## Output

There should be one line of output, containing the string "Yes" if the code table is ambiguous for the given message, or "No" otherwise.

| **Input Samples** | **Output Samples** |
| --- | --- |
| 3 0 s 1 a 00 t sat | Yes |

| 3 01 s 11 a 00 t sat | No |
| --- | --- |

# Insane Lock

By Francisco Elio Parente Arcos Filho, UEA BR Brazil

**Timelimit: 1 (SAP 9)**

Farcos wants to test the security of his new lock and for that he needs to find out how many ways it is possible to unlock it.

The padlock lock consists of two concentric rings, one fixed and the other mobile, with **N** positions each. All ring positions contain a number which, after microscopic analysis of a disassembled padlock, he found to be associated with the internal structure of the position.

When rotating the mobile ring and aligning it with the fixed ring positions, the aligned position pairs adjust according to an internal structure of small rods and sockets. Such structures are present in both fixed and mobile ring positions.

Each position of either of the two rings has 31 ordered microstructures that can be either a small rod, or a small socket. If in the alignment of the rings the rod of one position is opposite to the socket of one position on the other ring, then they fit and form a connection. But if in opposite ring positions, two rods oppose each other, then both retract and this is called a collision. The retraction of a rod is undone in another alignment when opposed to a socket.

The value of each position in each ring is associated with the numbering of its internal microstructures. From left to right they are numbered from 0. But only if the **i**-th microstructure is a rod does it add to the position value with 2**i**. For example, if microstructures 1, 3 and 4 of that position are rods and all the others are sockets, then the position value is 26 (21 + 23 + 24).

To unlock the lock it is necessary to make exact **K** rotational movements and at the end of the last one, all pairs of aligned positions must present the same pattern of connections, that is, if in a pair of aligned positions there is a connection in the microstructure of number **J**, all aligned pairs must have a connection in microstructure **J**. No matter if it is from fixed ring to mobile or the other way around.

Note that a rotation movement is a displacement of the mobile ring in **P** positions and in a single direction, either clockwise or counterclockwise, where 0 < **P** < **N**. The first rotation is always clockwise and thereafter it alternates between counterclockwise and clockwise.

It is worth to note that an alignment of the positions always implies an alignment of its internal microstructures which had the same number.

## Input

The first line of the input contains two integers **N**(2 ≤ **N** ≤ 106) and **K** (1 ≤ **K** ≤ 109), the number of positions in a ring and the number of rotations required to unlock the lock.

The second line contains **N** integers **Fi** (0 ≤ **Fi** < 231) separated by blank space representing the fixed ring positions.

The third line contains **N** integers **Mi** (0 ≤ **Mi** < 231) separated by blank space representing the positions of the mobile ring.

The respective positions on the two lines correspond to the initial alignment, that is, the first given number of the fixed ring is aligned with the first given number of the mobile ring, the second with the second and so on.

## Output

The output contains a single integer representing the number of ways to unlock the lock. Since this number can be very high, print only your remainder of division by 109+7.

| **Input Samples** | **Output Samples** |
| --- | --- |
| 3 1  12 9 12  10 15 15 | 1 |

| 4 2  11 11 11 11  5 5 5 5 | 9 |
| --- | --- |

| 4 2  3 8 2 9  7 12 6 13 | 5 |
| --- | --- |

| 4 5  3 8 2 9  7 12 6 13 | 121 |
| --- | --- |

# Parkour

By Francisco Elio Parente Arcos Filho, UEA BR Brazil

**Timelimit: 1 (SAP 11)**

A parkour practitioner intends to make a video with several maneuvers where he will jump between several buildings in an ever-increasing trajectory, that is, he will always jump from one building to another that has greater height.

Although dangerous is something that attracts attention in sport, this sportsman does not intend to take any more risks than necessary.He intends to make his way from the building he chooses to start to the largest building he can ever jump to the nearest option on the left or right.

In other words, since **Hi** is the height of **i**-th building from the beginning of the street, he can jump from building **i** to building **j** if **Hi** < **Hj** and there is no building **k** such that **Hi** < **Hk** and **i** < **k** < **j** or **j** < **k** < **i**.

Your task is to determine how many different paths there are to make this video.

The description of the street where it will be filmed, the height of each building and its distance from the beginning of the street will be provided.

The traceur (person who practices parkour) can start in any of the buildings.

## Input

The first line of input consists of a integer **N** (1 ≤ **N** ≤ 106), the number os buildings in the street.

The following **N** lines each consists of two integers: **H** (1 ≤ **H** ≤ 109) and **X** (1 ≤ **X** ≤ 109), that indicate the height of a building and its distance from the beginning of the street, respectively.

## Output

The output must be a single line containing the number of ways to trace the path to the parkour video. Because this number can be very large, print only your module 109+7.

| **Input Samples** | **Output Samples** |
| --- | --- |
| 4 31 30 63 10 127 130 15 70 | 7 |

| 6 17 6 35 9 30 1 19 3 18 8 20 4 | 15 |
| --- | --- |

| 9 3 1 5 2 1 3 6 4 3 5 4 6 7 7 2 8 4 9 | 14 |
| --- | --- |

# Roads

By Giovanna Kobus Conrado, University of SÃ£o Paulo BR Brazil

**Timelimit: 1 (SAP 12)**

Tland is a very beautiful country. It consists of **N** cities connected by **N**-1 bidirectional roads in such a way that there is a path from every city to very other city. Because of a storm the capitol is currently facing a really heavy shortage of food.

All the other cities offered to send a truck full of food to help, but there is one problem: each of the roads in Tland can only handle so much weight at a time.

So now the government of Tland wants to know: for each city in Tland what is the heaviest truck they can send to the capitol such that it does not exceed the weight limit for any road it will have to go through.

## Input

The first line of input consists of an integer **N**, the number of cities in Tland (1≤**N**≤105).

The following **N**-1 lines each consist of three integers: **V**, **U** and **C**, that indicate that there is a road from city **V**, to city **U** that can handle trucks that weigh at most **C** quilograms (1≤**V**,**U**≤**N** and 1≤**C**≤109).

The capitol will always be the city represented with **N**.

## Output

The output must be a line containing **N**-1 integers: the **i**-th of them indicates the weight of the heaviest truck that city **i** can send to the capitol.

| **Input Sample** | **Output Sample** |
| --- | --- |
| 5 3 5 3 2 4 1 3 2 4 1 4 1 | 1 3 3 1 |

# Impossible Followers

By Daniel Bossle, UFRGS BR Brazil

**Timelimit: 2 (SAP 12)**

Bini loves to buy products from a very famous website called AquiLogress. The site offers everything, even things you think you didn't need, but after you find out, you end up buying. Bini is one of them. In his most recent purchases, Bini bought a machine that claimed to be able to prove that **P != NP**. But as she got stuck in customs, he didn't even find out if that was true or not.

Wanting to take some extra money out of the pandemic (to be able to buy more "muambas"), Bini decided to open an open-box channel for products on a well-known social network. But the idea didn't work out very well: people didn't seem to be interested in items that he bought because he didn't say how much he spent on purchases. After a careful research, Bini found that if among **P** products in the store he chooses a value **Vi** among a range [**Ai**, **Bi**] of possible values, with **i** the as **i**-th product, having ∑Pi=1Vi2∑i=1PVi2 maximum and ∑Pi=1Vi∑i=1PVi equal to **N**, he will be able to please his followers. To inflate his spending on social media, Bini wants to show that he spent ∑Pi=1Vi2∑i=1PVi2 instead of ∑Pi=1Vi∑i=1PVi. Bini doesn't know how to program and asked you for help. Can you help Bini solve this problem?

## Input

The first line of each test case consists of two integers, **P** (1 ≤ **P** ≤ 102) and **N** (1 ≤ **N** ≤ 104). The next two lines are composed of **P** integers each, the first with the values ​​**Ai** (0 ≤ **Ai** ≤ 104) for each product **i** and the second with the values ​​**Bi** (0 ≤ **Bi**≤ 104) for each product **i**.

## Output

For each test case, your program must print a single number, containing the maximum amount that Bini will spend on the purchase of the "muambas". If there is no answer, your program should print the word **"IMPOSSIBLE"**, without quotes.

| **Input Sample** | **Output Sample** |
| --- | --- |
| 2 8 3 2 4 6 | 34 |

|  |  |
| --- | --- |
| 1 5 3 4 | IMPOSSIBLE |

|  |  |
| --- | --- |
| 3 10 3 2 2 5 4 5 | 38 |