National Olympiad in Informatics

Finals Round 1

Important! Read the following:

Hidden Test Cases. Your solution will be checked by running it against one or more (usually several) hidden test cases. You will not have access to these cases, but a correct solution is expected to handle them correctly.

Strict Output Format. The output checker is **strict**. Follow these guidelines strictly:

- It is **space sensitive**. Do not output extra leading or trailing spaces. Do not output extra blank lines unless explicitly stated.
- It is **case sensitive**. So, for example, if the problem asks for the output in lowercase, follow it.
- Do not print any tabs. (No tabs will be required in the output.)
- Do not output anything else aside from what's asked for in the Output section. So, do not print things like "`Please enter t`".

Not following the output format strictly and exactly will likely result in the verdict "*Output isn't correct*".

Use Standard I/O. Do not read from, or write to, a file. You must read from the standard input and write to the standard output.

Submit Code Only. Only include **one** file when submitting: the source code (.cpp, .py, etc.) and nothing else.

No Java Package. For Java submissions, do not include a `<MINTED>` line.

No Weird Filenames. Only use letters, digits and underscores in your filename. Do not use spaces or other special symbols.

Use Fast I/O. Many problems have large input file sizes, so use fast I/O. For example:

- In C/C++, use `<MINTED>` and `<MINTED>`.
- In Python, use `<MINTED>`

Flush On Interactive Problems. On interactive problems, make sure to **flush** your output stream after printing.

- In C++, use `<MINTED>` or `<MINTED>`
- In Python, use `<MINTED>` or `<MINTED>`
- For more details, including for other languages, ask a question/clarification through CMS.

Good luck and enjoy the contest! 😃

Contents

Notes

- Many problems have large input file sizes, so use fast I/O. For example:
  - In C/C++, use `<MINTED>` and `<MINTED>`.
  - In Python, use `<MINTED>`
- On interactive problems, make sure to **flush** your output stream after printing.
  - In C++, use `<MINTED>` or `<MINTED>`
  - In Python, use `<MINTED>` or `<MINTED>`
  - For more details, including for other languages, ask a question/clarification through CMS.

Good luck and enjoy the problems!

Problem A
Method of Least Squares

In statistics, there is a well-known technique called the Method of Least Squares. The technique is used for solving a problem in statistics called "data fitting". This technique was developed in the early 1800s by the famous mathematicians Legendre, Gauss, and Laplace.

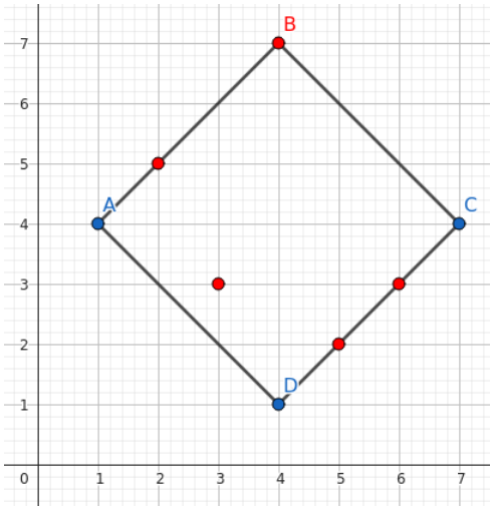
For this problem, you are asked to recreate decades of work from these great mathematicians in the matter of a few hours.

You are given n "data points", which are literally points on a 2D plane. Your task is to find the smallest square in terms of area (also known as the *least square*) that can "fit" all of these data points. In other words, all the data points must lie either inside the square, or on the edges, or on one of the corners.

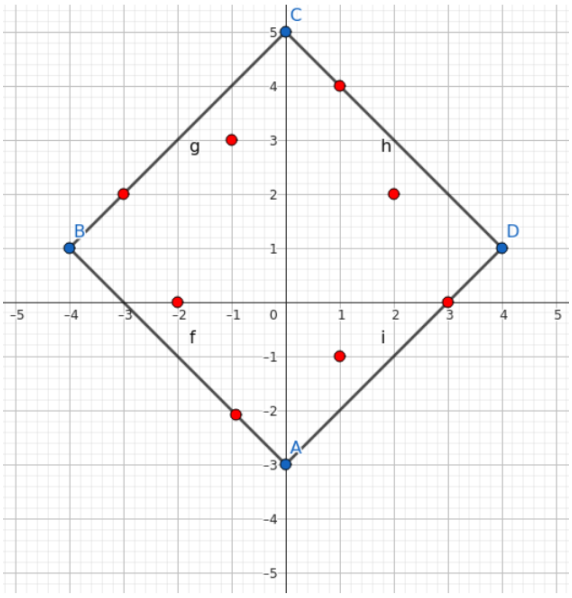
As an additional challenge, statisticians are concerned with creating "nice" models for their data sets. So, among all possible squares you can choose, you may only choose ones that satisfy *both* of the following criteria:

- It is possible to choose two of the square's corners such that those points share the same x coordinate.
- It is possible to choose two of the square's corners such that those points share the same y coordinate.

Here are examples of least squares, among squares that satisfy the above "niceness" condition. The data points are marked in red, and the corners of the least square are labeled $ABCD$.



Here, corners B and D share the same x-coordinate, while corners A and C share the same y-coordinate. The side length of this square is $3\sqrt{2} \approx 4.24264$, and its area is 18.



The side length of *this* square is $4\sqrt{2} \approx 5.65685$, and its area is 32.

Given our data points, please output the **minimum area** among all squares that: "fits" those data points; and is "nice".

Input Format

The first line of input contains the single integer n.

Then, n lines follow, each containing two space-separated integers x and y, meaning the point (x, y) is in our data set.

Output Format

Output a single decimal value, the area of the least square that is nice.

Your answer will be accepted if it has an absolute or relative error of at most 10^{-6} from the judge's answer. In symbols, let ans_{you} be your answer, and let ans_{judge} be the judge's answer. Your answer will be accepted if

$$|ans_{\text{you}} - ans_{\text{judge}}| \leq 10^{-6}$$

In short, **make sure you print your answer to sufficiently many decimal places**. For example,

- In C++, <MINTED> the <MINTED> library, and use

  <MINTED>

  to output <MINTED> to exactly 12 decimal places.
- In Python, use <MINTED> to output <MINTED> to exactly 12 decimal places.

Constraints

For all subtasks
$2 \leq n \leq 2 \times 10^5$ $\lvert x\rvert, \lvert y\rvert \leq 10^7$ for each data point. The data points are unique.

Subtask	Points	Constraints
1	**27**	$n = 2$
2	**21**	All points lie on the positive x-axis.
3	**23**	Each point either: lies on the positive x-axis, or lies on the positive y-axis.
4	**29**	No further constraints.

Sample I/O

Input 1	Output 1
5 2 5 3 3 4 7 5 2 6 3	18.000000

Input 2	Output 2
8 -3 2 -2 0 -1 -2 -1 3 1 -1 1 4 2 2 3 0	32.000000

Problem B
Astig Runnings

The biathlon is a popular winter sport, combining cross-country skiing and rifle shooting. With the Philippines being a tropical country, if we wanted to join, we would definitely be at a disadvantage. However, inspired by the Jamaican Bobsleigh Team (and the Philippines' own history of sending alpine skiers to the Winter Olympics!), the PH Biathlon Team dares to dream.

The Philippine Biathlon Team has asked NOI.PH to help make an interesting training course for them, and so the following game was proposed.

There are m barriers lined up in a row.

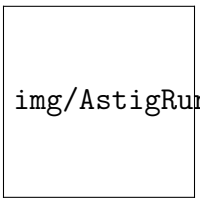
A biathlete is stationed at the far left. A target is stationed at the far right. All m barriers lie between the contestant and the target.

Each barrier can be either "closed" or "open". Contestants can see the starting configuration.

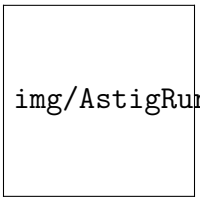
The contestant must fire their rifle in the direction of the target. But the barriers may be in the way!

- *If the bullet reaches an open barrier, it successfully passes through. The air pressure of the bullet then immediately causes the barrier to close behind it.*
- *If the bullet reaches a closed barrier, it strikes it! The bullet's path stops here, but the barrier becomes open as a result of the impact*

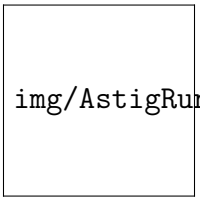
For example, consider the following configuration of barriers:



If they shoot the rifle once towards the target, here's what happens:



This would be the ultimate state of the barriers after that one shot:



The biathlete cannot hit the target until they manage to configure the barriers such that all of them are open before they take their shot (and the only way they can toggle this configuration... is also by just repeatedly firing their rifle). The contestant may fire their rifle multiple times, but they are limited by the number of bullets they have on them upon starting this course.

A contestant wins the game if they have enough bullets so that they can shoot the target.

This is a perfectly fine game, but it's just missing that NOI.PH *je ne sais quoi*. Of course we have to add a twisty twist. And what better way than to inject some randomness into the game?

We begin by constructing a *reference arena* with n barriers, labeled 1 through n from left to right, each either open or closed. When a biathlete arrives, we do the following:

- There are $n(n+1)/2$ pairs of indices (ℓ, r) such that $1 \leq \ell \leq r \leq n$; one of these is selected uniformly at random.
- Place them to the left of barrier ℓ, and the target to the right of barrier r.
- The biathlete plays the game from here, trying to shoot the target from their position (the barriers ℓ through r, inclusive, lie between the biathlete and the target; this means there are $m := r - \ell + 1$ barriers between them).
- Win or lose, after the game, barriers ℓ through r are **reset** to their original configuration.

The biathletes played our game q times. You are told how many bullets the playing biathlete had with them when they started each attempt. You are reminded that each attempt is independent from the others, since the barriers' configurations are restored when the game is over.

Help us determine, for each attempt, the probability that the biathlete wins. Specifically, for each attempt, just count the number of pairs of indices (ℓ, r) such that if this (ℓ, r) were chosen, then the biathlete would be able to shoot the target from those positions (given the number of bullets they have left).

Oh... So they're- This is just the skiing and shooting thing? I see... When you said you needed my help entertaining a group of rowdy biathletes, I thought you meant-

Input Format

The first line of input contains the two space-separated integers n and q.

The second line of input contains a string of length n, encoding the initial state of the barriers. The ith character is `O` if the ith barrier from the left is open, and `C` if it is closed.

This is followed by q lines, each describing an attempt by some biathlete. The t'th of these lines contains a positive integer b_t, the number of bullets the biathlete has in the t'th described attempt.

Output Format

Output q lines, the t'th of which should contain the answer for a biathlete who has b_t bullets.

Constraints

For all subtasks
$1 \leq n \leq 2 \times 10^5$ $1 \leq q \leq 3 \times 10^5$ $1 \leq b_t \leq 10^{18}$ for each t.

Subtask	Points	Constraints
1	**7**	$b_t = 1$ for all t
2	**6**	$b_t \leq 2$ for all t
3	**20**	All barriers in the reference arena are closed.
4	**18**	$n \leq 50$ $q \leq 50$
5	**24**	$n \leq 50000$ $q \leq 50$
6	**25**	No further constraints.

Sample I/O

Input 1	Output 1
7 3 OOCOCCC 3 2 21	12 9 23

Problem C
Electra Boom

We're NOI.PH guys, of course we think ***interactive*** *is the best type of problem!*

Electra Boom is one of my favorite YouTubers! She works with circuitry and electronics, and she makes it approachable by being unafraid to show things going wrong (and usually when they go wrong, they go *BOOM!*).

In one of her series, she buys shady electronics parts on the internet and tests them to see how badly they malfunction. The more catastrophic the failure, the higher she ranks it on the *B💣OM METER*!

For this week's episode, she has acquired an ominous-looking blackbox from the website Teμ. This black box has n **input nodes** on top, and n **output nodes** at the bottom, each arranged in a row and labeled 1 to n from left to right. There are n wires of negligible width, connecting each of the input nodes with a different output node.

In a somewhat-concerning design choice, the wires are bare and have no form of insulation whatsoever. Also, the wires are pressed flat together when the blackbox is shut. This means that if two wires have to cross over each other in some configuration, then there *will* be a point where those two wires make direct contact! At the very least, due to how the nodes were spaced apart, it is given that it is impossible for three wires to ever meet at a single point.

For an example, consider the configuration illustrated in the following diagram. The points of contact have all been highlighted.

If Electra Boom runs a live current through this blackbox, each of those points of contact makes a *BOOM!* Precisely, she hears one *BOOM* for every pair of wires that make contact in the blackbox's current configuration (and she is able to precisely count the number of *BOOMs* by performing a fourier transform on the waveform of the audio), and this number serves as the product's ranking on the *B💣OM METER*.

This week's video is special... because it's not a video, it's a livestream! She's going to be reading chat for suggestions on how she should tinker with the blackbox.

Due to the sealed nature of the blackbox, this is the only operation available to Electra Boom:

- Choose two output nodes i and j, and swap the wires connected to those endpoints.
    - Formally, the wire that was connected to output node j is now connected to output node i, and vice versa.

It is guaranteed that the wires will not "snag" on each other after each swap—that is, each wire always remains a straight line connecting its two endpoints.

Electra Boom (irresponsibly) runs live current through the blackbox, and reports to her audience the blackbox's rating on the *B💣OM METER*. She also (still irresponsibly) runs live current through the blackbox after *each* operation, and then reports the rating on the *B💣OM METER* after each operation.

For example, in the diagram from earlier, Electra Boom would report the blackbox as getting a 6 on the *B💣OM METER*. After swapping the wires connected to output nodes 1 and 3, it looks like this. Electra Boom would now report that the blackbox gets a 3 on the *B💣OM METER*.

It's your chance to keep Electra Boom safe by bringing the blackbox's rating on the *B💣OM METER* down to 0. And for the sake of her wellbeing, please try your best to do so in as few operations as you can manage.

We're NOI.PH guys, of course we like problem statements with nonsensical stories!

We're NOI.PH guys, of course we enjoy squeezing points out of problems with special scoring formulas!

Interaction

The interactor first sends an integer T, the number of test cases. Each test case proceeds as follows.

The interactor first sends a line containing the two space-separated integers n and I—the number of nodes, and the blackbox's initial rating on the *B💣OM METER* when it is first plugged in.

To tell Electra Boom what operation to do, print a line containing two space-separated integers i and j (where $1 \leq i, j \leq n$, and $i \neq j$), denoting the output node endpoints whose wires you want to swap.

When printing this line, **be sure to flush the output** so that Electra Boom can see your comment in the chat. Otherwise, Electra Boom might not receive your

output, and she will wait forever.

- In C++, use `<MINTED>` to flush the stream, or `<MINTED>` to print a newline character and then flush the stream.
- In Python, use `<MINTED>` to flush the stream, or use `<MINTED>` to print a newline character and then flush the stream.
- For more details, including for other languages, ask a question/clarification through CMS.

After that, the interactor responds by sending a line containing a single integer B.

- If $B = -1$, this means that you have either made an invalid output or exceeded the number of allowed operations. End the interaction to obtain a Wrong Answer verdict.
- Otherwise, B is the number of *BOOMs* that Electra Boom heard after performing your proposed operation.
  - In particular, if $B = 0$, then the blackbox has been neutralized (i.e. no *BOOM*s were heard). **When this happens, the next test case immediately begins.**
  - On the other hand, if $B > 0$, suggest another operation.

You can make at most 10000 operations in a single test case.

Constraints

For all subtasks
$1 \leq T \leq 20$ $I > 0$

Subtask	Points	Constraints
1	**15**	$n \leq 7$
2	**25**	$n \leq 99$
3	**60**	$n \leq 300$

There is also partial scoring for subtasks 2 and 3.

For each subtask, if you got a -1 response in any test case in any test file under that subtask, you get 0 points for that subtask.

Otherwise, let Q be the maximum number of queries made in any single test case, across all test files under that subtask.

- If you pass all test files under subtask 1, your score is 15.
- If you pass all test files under subtask 2, your score is $\min\left(25,\ 25 \cdot \left(\frac{5000}{Q}\right)\right)$.
- If you pass all test files under subtask 3, your score is $\min\left(60,\ 12 \cdot \log_2\left(\frac{20000}{Q}\right)\right)$.

Sample Interaction

The blank lines have only been added to illustrate the chronology of the back-and-forth between the submission and the interactor.

```
Input       Output
2
3 1
            1 2
2
            2 3
3
            3 1
0
2 1
            1 2
0
```

Testing Tool

A testing tool written in Python 3 is provided in the attachments in CMS to help with local testing.

- `interactive_runner.py` is a special program that pipes the input/output of your solution with the output/input of the interactor.
- `testing_tool.py` serves as the interactor, processing the logic that the judge is responsible for.
- `input.txt` is where we store the values of the parameters (the "input" of the interactor) that we would like to test our solution against. In this case, it contains the initial configuration of the blackbox.

The format of "`input.txt`" is as follows. A sample `input.txt` file is also included as an attachment.

The first line of the file contains a single integer T. This should be followed by the descriptions of the T test cases.

The first line of each test case should contain the integer n. Since there are n wires, suppose we arbitrarily label those wires $1, 2, 3, \ldots, n$.

The second line of each test case should contain n space-separated integers, where the ith integer is the label of the wire connected to input node i. No label should appear more than once.

Similarly, the third line of each test case should also contain n space-separated integers, where the ith integer is the label of the wire connected to *output* node i. No label should appear more than once.

For example, the test case

```
5
3 1 2 5 4
4 1 3 5 2
```

in `input.txt` would mean:

- Input node 2 and output node 2 are connected by the wire labeled 1
- Input node 3 and output node 5 are connected by the wire labeled 2
- Input node 1 and output node 3 are connected by the wire labeled 3
- Input node 5 and output node 1 are connected by the wire labeled 4
- Input node 4 and output node 4 are connected by the wire labeled 5

To run the testing tool, in your terminal, run

```
python3 interactive_runner.py python3 testing_tool.py
    input.txt -- <COMMAND>
```

where `<COMMAND>` would be the entire command you use to run your solution, such as

- "`python3 myFile.py`" for Python, or
- "`./executable`" for C++ (note that this is the compiled program, not the source code)

For more verbose output (which might help in debugging), you may add an integer from 0 to 2 before the `--` in the command (where 0 is the default, and 2 is maximum verbosity).

So, for example,

```
python3 interactive_runner.py python3 testing_tool.py
    input.txt 1 -- python3 myFile.py
```

If your solution would be marked as Wrong Answer, the testing tool will output an appropriate message. Otherwise, it will output "`CORRECT`".

Problem D
The Manga Guide to Algorithms

After many years, NOI.PH has finally released another entry in the Manga Guide series by Saji Tan—this is The Manga Guide to Algorithms! We hope that this and our many other outreach efforts help competitive programming reach an ever wider audience.

The scene has grown so much over the years, and we hope that it will continue to grow and prosper in the years to come.

To celebrate, NOI.PH and Saji Tan have decided to organize a book-launching event for the new Manga Guide, with free food afterwards for all invited guests (catered by a celebrity chef)! Help us decide the menu, taking into account people's preferences, but also keeping in mind any allergies.

There are n guests attending the event, labeled 1 through n. There are 10^9 possible types of ingredients in the world that our celebrity chef can use, labeled 1 through 10^9. A dish is assembled by choosing some **non-empty** subset of these ingredients to put in that dish.

Prior to the event, we collected m *opinions* from the guests. Each opinion is in one of two forms:

- "Guest i **must have** ingredient t"; or
- "Guest i **cannot eat** ingredient t".

To keep the menu simple, our celebrity chef will only put **two** dishes on the menu. Please determine if it is possible to make it such that each person has a dish that they would love to eat (that is, each person's meal should have all the ingredients they must have, but none of the ingredients they cannot eat).

Also, we will be preparing packed meals for the event, so we would have to determine ahead of time which dish to make for each guest.

If yes, please also tell us exactly what to do! That, is, output three things:

- Which ingredients to put in the first dish.
- Which ingredients to put in the second dish.
- An assignment of, for each of the n guests, which of the two dishes will be made for them.

Among all possible assignments, it is also preferable for the number of guests eating each dish to be *as balanced as possible*. That is, let A be the number of guests assigned to eat the first dish, and let B be the number of guests assigned to eat the second dish. We would *prefer* if $|A - B|$ is minimized. If not, but the answer would otherwise be valid, then you can still earn partial points.

Input Format

The first line of input contains two space-separated integers n and m.

Then, m lines follow, each describing an opinion. Each line consists of three space-separated tokens, each one either:

- "i `:)` t" or
- "i `>:(` t"

meaning that guest i loves (or hates) ingredient t, respectively.

Output Format

First, output a line containing the token `:)` or `>:(`, depending on whether the task is possible or not (respectively).

If `:)`, please also output the following, each in their own line:

- First, a positive integer k_A, the number of ingredients to put in the first dish.
- Then, k_A distinct space-separated integers, each from 1 to 10^9—the ingredients to put in the first dish.

- Then, a positive integer k_B, the number of ingredients to put in the second dish.
- Then, k_B distinct space-separated integers, each from 1 to 10^9—the ingredients to put in the second dish.
- Finally, n space-separated integers, each one either 1 or 2. The ith integer represents which meal should be eaten by the ith guest (1 for the first dish, and 2 for the second dish).

Each of k_A and k_B should be at least 1 and at most 4×10^5. If there are multiple possible solutions, any will be accepted (and scored accordingly).

Constraints

For all subtasks
$2 \leq n \leq 2 \times 10^5$ $0 \leq m \leq 3 \times 10^5$ $1 \leq$ each ingredient number $\leq 10^9$ No guest says their opinion about the same ingredient twice No guest both likes and dislikes the same ingredient.

Subtask	Points	Constraints
1	**15**	$n \leq 10$ $m \leq 2000$
2	**35**	$n \leq 5000$
3	**30**	Each ingredient is mentioned in no more than two opinions.
4	**20**	No further constraints.

There is also partial scoring:

- You get 0 points for this subtask if there exists a test file under this subtask where either of the following holds:
  - Your `:)` or `>:(` answer was incorrect.
  - For some `:)` solution, your construction was invalid or incorrect.
- If not, you get 60% of the points for this subtask if there exists a test file under this subtask such that: your construction was valid, but did not minimize $|A - B|$.
- Otherwise, you get 100% of the points for this subtask.

Sample I/O

Input 1	Output 1
8 9 1 :) 123 2 :) 123 3 :) 123 4 >:(123 4 >:(456 5 >:(456 6 :) 456 6 >:(789 7 :) 789	:) 3 123 456 1000000000 4 789 404 420 1000000000 1 1 1 2 2 1 2 2

Input 2	Output 2
3 6 1 :) 997 2 >:(997 1 >:(998 2 :) 998 3 :) 997 3 :) 998	>:(