IDI Open Programming Contest March 18th, 2023

Problem Set

Jury and Problem Writers

Jean Niklas L'orange (Head Judge)

Test Solvers

Bjarte Klyve Larsen
Johan Sokrates Wind
Martin Grønlien Pejcoch

Rules

TL;DR: Teams of up to three persons try to solve as many problems as possible from a set, without external help.

Before the contest begins, you are allowed to log in on your assigned computer, and log in on the submission system. You may do nothing else with the computer (such as starting to write code). You may not touch the problem set before the contest has started.

Contestants are *only* allowed to communicate with members of their own team, and the organisers of the contest. You may not surf the web (except for allowed content), read e-mail, chat on Slack, or similar things. The only network traffic you may generate is from submitting problem solutions, and access to content specified by the local organisers.

- What you may bring to the contest floor:
  - Any written material (Books, manuals, handwritten notes, printed notes, etc).
  - Pens, pencils, blank paper, stapler and other useful non-electronic office equipment.
  - NO material in electronic form (CDs, USB pen and so on).
  - NO electronic devices (Cellular phone, PDA and so on).
- What you may use during the contest:
  - What you brought to the contest floor (see above).
  - Your assigned (single) computer.
  - The specified system for submitting solutions: `https://idio23.kattis.com`
  - Printers designated by the organiser.
  - The external documentation for your language of choice. This is documented in the tutorial for your language at `https://idio23.kattis.com/help`
  - All compilers and IDEs pre-installed on your assigned computer
  - Non-programmable tools which are a natural part of the working environment (such as `diff` and `less`).

The problem set consists of a number of problems (usually 8-12). The problem set will be in English, and given to the participating teams when the contest begins. For each of these problems, you are to write a program supported by the Kattis system. The jury guarantees that each problem is solvable in C, C++, Java and Python 3. No guarantees for other languages are given due to the large number of allowed languages, however the jury guarantees that for every language there is at least one problem solvable in that language. It has always been the case that several of the problems were solvable in all available languages, but there is no guarantee of this.

Your programs should read from standard in and write to standard out. See `https://idio23.kattis.com/help` for a tutorial and the compiler options for your language of choice.

The team that solves the most problems correctly wins. If two teams solve the same number of problems, the one with the lowest total time wins. If two top teams end up with the same number of problems solved and the same total time, then the team with the lowest time on a single problem is ranked higher. If two teams solve the same number of problems, with the same total time, and the same time on all problems, it is a draw. The time for a given problem is the time from the beginning of the contest to the time when the first correct solution was submitted, plus 20 minutes for each incorrect submission of that problem. The total time is the sum of the times for all solved problems, meaning you will not get extra time for a problem you never submit a correct solution to.

If you think some problem is ambiguous or underspecified, you may ask the judges for a clarification request through the Kattis system. The most likely response is "No comment, read problem statement", indicating that the answer can be deduced by carefully reading the problem statement or by checking the sample test cases given in the problem, or that the answer to the question is simply irrelevant to solving the problem.

Input Validation

In general we are lenient with small formatting errors in the output, in particular whitespace errors within reason. But not printing any spaces at all (e.g. missing the space in the string "`1 2`" so that it becomes "`12`") is typically not accepted. The safest way to get accepted is to follow the output format exactly.

For problems with floating point output, we only require that your output is correct up to some error tolerance. For example, if the problem requires the output to be within either absolute or relative error of 10^{-4}, this means that

- If the correct answer is 0.05, any answer between 0.0499 and .0501 will be accepted.
- If the correct answer is 500, any answer between 499.95 and 500.05 will be accepted.

Any reasonable format for floating point numbers is acceptable. For instance, "`17.000000`", "`0.17e2`", and "`17`" are all acceptable ways of formatting the number 17. For the definition of reasonable, please use your common sense.

Tips

- Tear the problem set apart and share the problems among you.
- Problems are **not** sorted by difficulty.
- Try solving the easy problems first. Two problems in this set are tagged with "(Easy)" to help point you in the right direction.
- If your solution fails on a problem, you can print your program and debug it on paper while you let someone else work on a different problem on the computer.
- All problems are guaranteed to be solvable in C, C++, Java and Python 3
- Look at the scoreboard if you are unsure which problem to work on next.

An Atypical Auction

Problem ID: anatypicalauction

This year, the auctioneer Amadeus has found some very impressive items to sell in his yearly auction. This year's theme is "Original Algorithm Proofs", and he has found big hits like Sharir's linear time connected component algorithm, as well as some obscure Tarjan papers.

This year, the auction is in Algoland, where there are strict rules around auctions to keep them fair and accessible to everyone. There are two requirements for an Algoland auction:

- every participant has to bid on all items presented, and
- every participant can buy at most one item

This makes the auction ordering very important. To avoid people "gaming" the system, the auction order is hidden for the participants. In particular, the auction follows these exact steps:

1. All items are presented, in no particular order, on a website
2. An independent third-party corporation receives binding bids from every participant for every item
3. When all bids have been received, the auctioneer decides on an order in which the items are auctioned at (without knowing what the bids are)
4. The auctioneer receives information about items where is no winning bid, i.e. at least two participants have bid the highest amount. How they handle those cases is up to them.

This process went as usual until Amadeus picked the item order: Instead of telling Amadeus identical winning bids, the system informed Amadeus that every bid was unique, and **printed them out** for Amadeus to see. Clearly, this seems to be a debug setting they forgot to turn off, and Amadeus immediately reported the bug to the developers.

Even though the auction order is now fixed, Amadeus wonders if he could have gotten even higher prices if he knew this information before he picked the auction order.

Input

The first line contains two integers N and M, the number of items and the number of participants, respectively. Then follows N lines, each with M integers.

The integer $c_{i,j}$ is the jth element on row i and represents the bid participant j offers on item i.

Output

Assuming Amadeus' auction order is item 0 then item $1, \ldots, N-1, N$, output the difference between the maximum possible income and the income provided by Amadeus' auction order. Then, on the next line, output the optimal item order, separated by whitespace. If there are multiple possible answers, you may pick any of them.

Limits

- $0 < N \leq M \leq 300$
- $0 < c_{i,j} < 1\,000\,000$
- All $c_{i,j}$ are distinct

Sample Input 1	Sample Output 1
3 3 50 75 100 60 95 125 70 115 150	30 2 1 0

Sample Input 2

```
3 5
600 530 750 800 390
500 490 350 1200 310
1500 1250 1800 2000 1450
```

Sample Output 2

```
500
1 2 0
```

Sample Input 3

```
3 3
2 8 4
3 5 6
9 1 7
```

Sample Output 3

```
0
0 1 2
```

Bamboozled Boardwalk
Problem ID: bamboozledboardwalk

Picture by Daniel Seßler

Success! Benjamin and his bridge and boardwalk building company Bridgesmith has gotten the contract to improve accessibility to nature around the town of Wildeley. The area has many beautiful locations, but all are rather inaccessible as they are surrounded by a big bog. To make them easily accessible, the town's mayor has decided that it should be possible to reach all of these "islands" by walking over boardwalks. There are many possible ways to connect these islands, so Bridgesmith has sent in a lot of boardwalk alternatives to Wildeley's mayor.

Unfortunately, the recent recession has also impacted Wildeley, and not as many tourists can afford to go there anymore. While the initial plans were more grandiose, the mayor has decided that they will only pick the cheapest boardwalks to connect all the "islands", and no more. It will still be a good sum of money and work for years to come, but not as much as Bridgesmith hoped for.

Benjamin has an idea though. Since Bridgesmith's profits are purely derived from the cost of their projects, driving the total cost up will bring in more profits. And although they have delivered all the alternatives over to the mayor, they can say that one of the alternatives has a grave error and must be retracted. But to not raise any suspicion and reopen the bidding process, at most one such retraction can be done.

Benjamin isn't entirely sure which boardwalk they should retract, however, as there are a lot of these "islands", and Bridgesmith has proposed a lot of boardwalk alternatives. Can you help Benjamin find which boardwalk they should retract (if any)?

Input

The first line contains two integers V and B, the number of "islands" (including the mainland) and the number of boardwalk proposals Bridgesmith has submitted to the mayor. Then follows B lines, one per boardwalk proposal, each with three integers u_i, v_i and c_i each:

- u_i and v_i are the "bog islands" this boardwalk will connect
- c_i is the cost of building this boardwalk

Output

Output the maximal total cost of all the boardwalks Wildeley wants to contract, assuming Bridgesmith can retract at most 1 boardwalk alternative.

Limits

- $1 < V \leq 250$
- $0 \leq u_i, v_i < V$ and $u_i \neq v_i$
- $0 < c_i \leq 10\,000$
- $V - 1 \leq B \leq V^2$
- The graph formed by $G = (V, B)$ is connected.

Sample Input 1

```
5 6
0 1 10
0 2 3
0 3 8
0 4 9
1 2 6
3 4 4
```

Sample Output 1

```
28
```

Sample Input 2

```
4 6
1 0 2
1 3 8
1 2 4
0 2 3
3 1 7
2 3 5
```

Sample Output 2

```
12
```

Checkpoint Champion

Problem ID: checkpointchampion

Chester is the best orienteerer at IDI, being the undefeated champion for what seems like an eternity. His navigation skills mean he is able to find the control points almost immediately, and usually finds the optimal path to the next control point.

This is a problem, as many people feel disheartened whenever they first join the orienteering club at IDI. "There's no way I can get to that level," they think, and usually stop participating in competitions after a month or two.

The orienteering committee has therefore started to make a new type of sport which focuses more on speed and less on navigational skills: Line segment orienteering. By replacing the control points with line segments, they think they can increase the number of participants significantly. And they have decided to utilize GPS for what it's worth: Instead of using an e-card they "punch" at the station, their location will automatically check them in. This means the participant won't have to look hard after a control point, and they don't have to stop to punch their card either.

Of course, this changes the rules of orienteering somewhat: In line segment orienteering, every participant starts at the point $S = (s_x, s_y)$. They then have to touch the line segments $\overline{L_i R_i}$ for $i = 1, 2 \ldots N$ in order, and the fastest person to reach the last line segment wins. You are allowed to touch a line segment i before and after you have touched segment $i - 1$, but you still have to touch it again after you touch line segment $i - 1$.

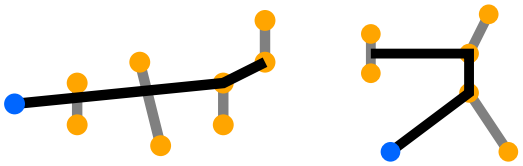


Figure 1: Solutions for sample inputs 1 and 2, respectively.

However, certain test runs reveal that the GPS is not as accurate as they want it to be: If a participant doesn't cross the line segment, the system doesn't always detect that the participant touches the segment. As the competitors are very competitive, this could be disastrous. To safeguard against this situation, they decide to ensure that the shortest path to the next line segment is always in front of the current line segment.

But now they have another problem: They don't know how to compute the course's length! Previously they used the straight line segments between the control points as a lower bound, but with these line segments, they would have to redo their computations. Can you help them?

Input

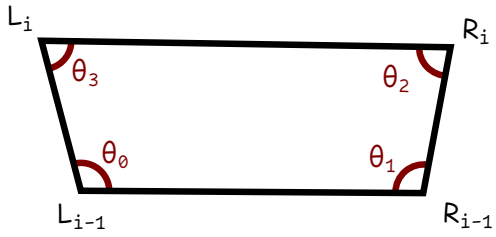
The first line contains three integers N, s_x and s_y, denoting the number of line segment checkpoints and the initial starting location $S = (s_x, s_y)$. Then follows N lines, each with 4 integers $l_{i,x}$, $l_{i,y}$, $r_{i,x}$, $r_{i,y}$, denoting the location of the ith checkpoint. The left point on the line segment is $L_i = (l_{i,x}, l_{i,y})$ and the right point on the line segment is $R_i = (r_{i,x}, r_{i,y})$.

Output

Output the distance of the shortest path from the starting location to the last line segment checkpoint, which passes through all the intermediate segments in order. Your answer must have an absolute or relative error of at most 10^{-6}.

Limits

- $0 < N \leq 5000$
- $-10\,000 < l_{i,x}, l_{i,y}, r_{i,x}, r_{i,y} < 10\,000$
- R_0 is right of $\overrightarrow{SL_0}$
- For all $1 < i \leq N$, let $A = L_{i-1}$, $B = R_{i-1}$, $C = R_i$ and $D = L_i$, be the vertices of the quadrilateral $\square ABCD$. Then, all the interior angles $0^\circ < \theta_0, \theta_1, \theta_2, \theta_3 < 180^\circ$:



Sample Input 1

```
4 -3 -1
0 0 0 -2
3 1 4 -3
7 0 7 -2
9 3 9 1
```

Sample Output 1

```
12.2859435986206797
```

Sample Input 2

```
3 0 0
4 3 6 0
4 5 5 7
-1 4 -1 6
```

Sample Output 2

```
12
```

Duck Division
Problem ID: duckdivision

Picture by U.S. Department of Agriculture

Dorothea's duck flock is getting a bit out of hand! The flock is now so big that it's very hard to handle them all, and getting them into the duck pen for the night takes almost the entire evening.

That's a problem, because Dorothea wants to watch the new series "Quacka" before going to bed. Now she doesn't have enough time to do that! However, she knows that herding a flock of D ducks into a duck pen take $\Theta(D^2)$ time. For that reason, she wants to temporarily split her flock into smaller *divisions* to make them easier to herd. They will all have their separate locations in her backyard, so there's no possibility of them mixing up and making it hard to herd them the following evening.

But ducks are very social animals, and a flock of ducks will be *sad* if there are fewer than U ducks in the flock. Dorothea wants to ensure that none of her ducks are sad. Additionally, Dorothea doesn't want to remember a lot of numbers, so the divisions must all be of the same size: This makes it much easier to check that none of the ducks have gone missing during the night.

Dorothea wonders if you could help her find the smallest size of a duck division that satisfies these criteria.

Input

The input is a single line with two integers, D and U, the number of ducks in Dorothea's flock, and the minimal number of ducks that should be in a division.

Output

Output the lowest acceptable size of a duck division.

Limits

- $0 < U \leq D \leq 10^{12}$

Sample Input 1	Sample Output 1
14 5	7

Sample Input 2	Sample Output 2
114 10	19

Sample Input 3	Sample Output 3
31 5	31

Estimated Endowment Elderliness

Problem ID: estimatedendowmentelderliness

After certain customer laws went into effect, all banks in Norway are now required to send you a transparent report on your fund accounts and the costs associated with them. Claudia was rather surprised when she got a letter from Euler Bank, telling her about a fund account named "For Dog" she didn't know she had!

After some digging around, she found out her grandparents put a lump sum into a fund in her name a long time ago. When Claudia was young, she really wanted a dog, and her grandparent thought it made sense to save up some money in a fund until she could take care of one herself. However, the age of the fund account is not present in the account report, and her grandparents don't remember exactly when they bought the fund shares.

The fund is one of the weirder ones over at Euler Bank: Its price is always monotonically increasing or decreasing during a month! Because her grandparent put in a lump sum and the capital has been in the same fund since it was bought, Claudia figures she can compute which month they bought the fund shares.

Euler Bank provides monthly rate of returns for all their funds, dating back to the fund's inception. A monthly rate of return r_i is defined as

$$r_i = \frac{C_i - C_{i-1}}{C_{i-1}}$$

$$C_i = \text{closing price on last day of month } i$$

For example, if $r_i = 1.23\%$ for February, then if Claudia had 100.00 NOK in the fund at the end of January, she would've had 101.23 NOK at the end of February. And if $r_i = -2.34\%$, she would've had 97.66 NOK instead.

The fund report tells Claudia, among other things, the total rate of return R_N for her fund shares. Can you help Claudia figure out how long ago the fund shares were bought?

Input

The first line contains an integer N and a percentage R_N, representing the number of complete months the fund has been available, and the total rate of return of the initial lump sum at the end of last month.

Then follows one line with N percentages r_i, separated by a single whitespace. r_1 is the fund's first monthly return and r_N is the monthly rate of return for the last month.

Percentages are written as a real number followed by a percentage sign.

Output

Output the number of years and months since the initial deposit in the format "`x year[s] and y month[s] ago`" (without the quotation marks).

If there are multiple possible answers, output the answer the furthest back in time. If there are none, output "`something fishy is going on`" (without the quotation marks).

Limits

- $0 < N < 250$
- Percentages in the input will have exactly 2 digits after the decimal point
- $-50.00\% < r_i < 100.00\%$
- $-100.00\% < R_N < 1\,000.00\%$
- The total rate of return for the fund account is never in the interval $[-0.01\%, 0.01\%]$ at the end of a month

Sample Input 1	Sample Output 1
2 10.00% 0.00% 20.00%	0 years and 1 month ago

Sample Input 2

```
5 9.60%
1.41%
1.62%
0.69%
2.72%
3.14%
```

Sample Output 2

```
0 years and 5 months ago
```

Sample Input 3

```
6 -10.00%
-20.00%
20.00%
-20.00%
20.00%
-20.00%
20.00%
```

Sample Output 3

```
0 years and 6 months ago
```

Sample Input 4

```
1 999.99%
3.00%
```

Sample Output 4

```
something fishy is going on
```

Sample Input 5

```
20 14.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
1.00%
```

Sample Output 5

```
1 year and 2 months ago
```

Fortunate Fees
Problem ID: fortunatefees

Picture by Kjetil Ree, CC BY-SA 2.5

Fred is an autonomous AI working for Predetermined Corp, a company that specialises in delivering goods between predetermined cities in Europe. Fred, for instance, is always driving from city u_0 to city v_0 and straight back. In fact, every autonomous AI i at Predetermined Corp drives from city u_i to v_i and back, and does nothing else.

This is pretty boring, because Predetermined Corp wants them to take the cheapest route possible every single time. As Predetermined Corp has struck an amazing deal with the EU, their entire fleet's energy and upkeep costs are completely subsidised, and the only real cost comes from the road tolls they have to pay.

Of course, it's not always obvious which route is the cheapest between two different cities. Fortunately for the AIs, IDI, the Intereuropean Department for Imposts, has a website with all the road tolls in Europe, making it trivial to compute the optimal route.

IDI is also responsible for deciding what the road tolls should be, and is about to change them once again. The process to do that is quite simple: For every road, they pick a random positive real number and let that be its road toll. This is quite the thrill for all the AIs at Predetermined Corp: They will still drive from u_i to v_i and back, but likely with a different route! Well, with the exception for the AIs that drive goods inside the same city ($u_i = v_i$), that is.

Fred knows that Predetermined Corp will never change an AI's schedule, but he and the other AIs still dream of exploring the world. With IDI's rather frequent changes to road tolls, they all wonder how many cities they will be able to visit given an infinite amount of time and an infinite amount of road toll changes.

Input

The first line contains two integers, C and T: The number of cities and the number of AIs at Predetermined Corp.

Next follows C lines. Each line starts with an integer N_i, followed by N_i intervals $I_{i,j} = [s_{i,j}, e_{i,j}]$. There is a two-way road between city i and $k \in I_{i,j}$.

Finally, T lines follow. Each line contains two integers u_i and v_i, the cities AI i drives between.

Output

Output T lines, one for each AI. For each line i, print out the number of cities AI i will eventually visit given an infinite amount of time.

Limits

- The cities and roads form a connected graph.
- $1 < C \leq 10\,000$
- $0 \leq s_{i,j} \leq e_{i,j} < i$ and $e_{i,j} + 1 < s_{i,j+1}$
- $0 \leq T \leq 2\,500$
- $0 \leq u_i, v_i < C$
- $\sum N_i < 20\,000$

Sample Input 1

```
7 2
0
0
1 [0,0]
2 [0,0] [2,2]
1 [3,3]
2 [1,1] [4,4]
2 [1,1] [5,5]
3 6
2 4
```

Sample Output 1

```
5
4
```

Sample Input 2

```
4 2
0
1 [0,0]
1 [0,1]
1 [0,0]
0 3
1 3
```

Sample Output 2

```
2
4
```

Sample Input 3

```
6 3
0
1 [0,0]
1 [1,1]
1 [0,1]
2 [0,1] [3,3]
2 [2,2] [4,4]
0 5
1 5
3 4
```

Sample Output 3

```
6
6
6
```

Gelatinous Gladiator
Problem ID: gelatinousgladiator


Artwork by Anindya Shiddhartha, mirrored

The small town of Gooville has gotten into a sticky situation. The only road to the outside world is through a narrow strait, and it has been infested with slimes!

Glenn, the village hero, has decided to step up and save the day. He will fight them all alone with his broadsword.

Now, contrary to what basically every RPG out there says, slimes aren't actually dangerous in the real world. But they are a nuisance, as they will still "attack" you. All slimes are of a specific size s, and whenever a size s slime "attacks" you, you end up being covered by u_s units of slime. While you're covered with slime, you can't really do anything but remove it, and it takes one second to remove one unit of slime.

To make matters worse, a group of slimes will all "attack" at the same time. Fortunately, for reasons not yet known to scientists, they will keep you alone if you are covered by any amount of slime. Perhaps they think you are one of them while you're covered in it?

Whatever the reason, Glenn has concocted a plan with this in mind. The plan is as follows:

1. Slice up to a_s slimes of one particular size s
2. Get "attacked" by all the remaining slimes
3. Remove all the slime
4. If any are left, goto 1

Whenever Glenn slices up a slime of size s, the slime splits into k_s slimes of size $s-1$. The only exception are slimes of size 1, which will always turn into inanimate puddles of slime.

But Glenn doesn't want to take too long: The town really loves garlic and ginger, and they will soon run out. Can you help Glenn clean up the slimes as fast as possible?

Input

The first line contains one integer S, the maximum size of a slime. Then follow S lines, one for each size of slime, starting from 1 going up. All of the lines contain four integers I_s, k_s, u_s and a_s:

- I_s is the number of initial s-sized slimes
- k_s is the number of $(s-1)$-sized slimes a s-sized slime will turn into when sliced up
- u_s is the amount of slime a single s-sized slime will cover Glenn with per "attack"
- a_s is the maximum amount of s-sized slimes Glenn can split in a single slice

Output

Assuming Glenn uses 1 second per slice, output the smallest time (in seconds) Glenn would be able to remove all the slimes.

Limits

- $0 < S < 5$
- $0 \leq I_i \leq i$
- $k_1 = 0$, and for $1 < i$, $0 \leq k_i \leq i$
- $0 \leq u_i \leq 100$
- $1 \leq a_i \leq 100$

Sample Explanation

The optimal choice for Glenn in the first sample is to take each turn as follows:

1. Slice the slime of size 3. It will split into two, and both will spew 3 units of slime on Glenn, making this turn take a total of $1 + 2 \times 3 = 7$ seconds.
2. Next, Glenn slices the two slimes of size 2 down to four slimes of size 1. They will all spew 2 units of slime on Glenn, making this turn take $1 + 4 \times 2 = 9$ seconds.
3. Glenn will now finish off as many slimes of size 1. He can remove at most 3, meaning there is only one left. This turn takes $1 + 1 \times 2 = 3$ seconds.
4. At the last turn, Glenn finishes off the remaining slime for the time it takes to slice, 1 second.

Sample Input 1

```
3
0 0 2 3
0 2 3 3
1 2 5 3
```

Sample Output 1

```
20
```

Sample Input 2

```
3
1 0 1 2
2 2 3 2
1 2 5 1
```

Sample Output 2

```
48
```

Halting Helicopter

Problem ID: haltinghelicopter

Whoah, that was quite the scare! Harjawaldar, a helicopter test pilot, did a miraculous save and managed to land the prototype helicopter with only minor damages.

Today's test run was for the helicopter producer Beeswax Corp, which needed help verifying that their new prototype could reach its expected altitude. The entire helicopter stopped reacting to any input after a certain height, causing the helicopter to quickly lose both height and a bit of stability. Fortunately, Harjawaldar had prepared for that outcome and managed to regain enough stability and lift to avoid the worst.

Beeswax Corp was of course saddened by the results, but was happy that Harjawaldar could provide them with the data they needed. However, while it's none of Harjawaldar's beeswax, he experienced a similar incident in the past, so he dared to ask their CTO Ikaros a simple question:

> "How'd you solve the icing problem?"
> "Icing problem?" Ikaros replied

It turns out Ikaros is somewhat surprised that cold temperatures were a problem at high altitudes, and not heat. This isn't surprising for the rest of the team, though they're interested in the exact altitude where Harjawaldar experienced the system malfunction. Trouble is, they forgot to install an altimeter in the helicopter!

Fortunately, it is possible to dig this information out of the helicopter "crash" log, though it's not easy. The altitude at any specific time t is defined as

$$h(t) = \left(\sum_{i=1}^{N-1} c_i t^i \right) - c_N t^N$$

Higher-order polynomials aren't Harjawaldar's strong suit, so he has called you for help. Can you find the time $h(t)$ is the highest? Harjawaldar is able to compute $h(t)$ himself, as long as he knows what t is.

Input

The first line contains a single integer N, the degree of the expression. On the next line, N integers $c_1, \ldots c_N$ follow, representing the coefficients for the terms $c_i t^i$.

Output

Output the value $0 \leq t$ which maximises $h(t)$. Your answer must have an absolute or relative error of at most 10^{-8}.

Limits

- $1 < N \leq 10$
- $0 < c_i \leq 100$

Sample Input 1	Sample Output 1
2 1 1	0.5

Sample Input 2	Sample Output 2
3 1 2 3	0.62283903

Imprisoned Imps

Problem ID: imprisonedimps

Cordelia is playing a farming game with a 2D grid, where you are a witch planting herbs. As you progress through the game, you can summon imps that aid you during the day, picking herbs and watering them. However, during the night, the imps tend to become nasty rascals that vandalise your property. To protect your crops, you'll have to contain them in some way.

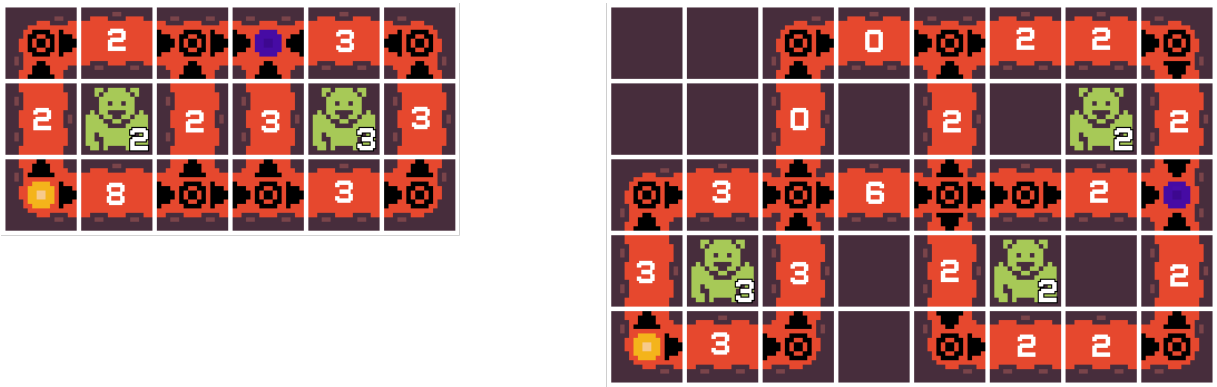


Figure 1: Sample inputs 1 and 2, with valid solutions. Tiles from 1-Bit Pack by Kenney

One way to contain them is by imprisoning them in a *lava network*. A lava network is a directed acyclic graph of lava pumps connected by straight lava channels. The first lava pump pumps lava from the Earth's core (the yellow dots in Figure 1) and the last lava pump pumps lava back down into Earth's core (purple dots). The remaining ones ensure the flow keeps going, and can distribute the incoming lava into multiple flows with varying flow rates if so desired.

Because the game is grid-based, the lava channels will always be axis-aligned. The grid also impacts the imps' movement: An imp can only move from its current grid tile to all adjacent grid tiles, though not diagonally. And an imp moving around can be a problem, because they know how to cast frost spells. If only one manages to move onto a weak spot in the lava network, they can solidify the entire thing. If that were to happen, Cordelia's entire farm would surely be destroyed!

More specifically, imp i can solidify the entire network if they move on top of a lava pump or lava channel with a *total* flow that is strictly lower than its spell level S_i. They are unable to move onto a grid tile where a lava pump/channel has a total flow equal to or greater than their spell level.

To ensure she can imprison them all, she has *enclosed* and *isolated* them. An imp is *enclosed* if it can't move to the tile at $(-1, -1)$ without walking over a lava pump/channel, and it is *isolated* if it can't move to a tile with another imp without walking over a lava pump/channel.

However, Cordelia's not entirely sure how to figure out how fast the lava has to flow in the different sections of her lava network, so she is working on making a program to compute the lowest possible flow she can use to imprison the imps.

Input

The first line consists of two integers I and P, the number of imps and the number of lava pumps.

Next follow I lines, one for each imp. Each line contains three integers $I_{x,i}$, $I_{y,i}$ and S_i, denoting the imp's position and its *spell level*.

Then follow P lines, one for each lava pump p. Each line starts with three integers: $C_{x,i}$ and $C_{y,i}$, denoting the pump's position, and O_i, denoting the number of **outward** flows this pump has. Then, on the same line, follows O_i integers, $V_{i,0}, V_{i,1} \ldots V_{i,O_i-1}$. Lava pump i can pump lava in a straight line from itself to all lava pumps $V_{i,j}$.

p_0 is the first lava pump, pumping lava up, and p_{P-1} is the last, pushing lava down into Earth's core.

Output

Assuming there is no limit to a lava channel's flow, output the minimal required units of lava per second the first lava pump must pump up from the Earth's core.

Limits

- $0 < I < 1000, 3 < P < 1000$
- $0 \leq I_{x,i}, I_{y,i}, C_{x,i}, C_{y,i} \leq 1000$
- $0 < S_i \leq 25$
- For $i \neq j$, $(C_{x,i}, C_{y,i}) \neq (C_{x,j}, C_{y,j})$
- For $i < P - 1$, $0 < O_i < 5$, and $O_{P-1} = 0$
- $0 < V_{i,j} < P$ and $V_{i,j} \neq i$
- Lava pump i can pump lava in a straight line towards lava pump $V_{i,j}$. The line will be in one of the cardinal directions. All outward flows from a lava pump will be in different cardinal directions.
- All imps are *enclosed* and *isolated* as specified by the problem statement.
- The lava network is a directed acyclic graph where all segments can flow to the last pump.

Sample Input 1

```
2 8
1 1 2
4 1 3
0 0 2 1 4
2 0 2 2 5
3 0 2 3 7
5 0 1 6
0 2 1 5
2 2 1 7
5 2 1 7
3 2 0
```

Sample Output 1

```
10
```

Sample Input 2

```
3 12
1 1 3
5 1 2
6 3 2
0 0 2 1 7
2 0 1 6
4 2 3 8 3 4
5 2 1 11
4 4 1 10
2 4 1 4
2 2 2 5 2
0 2 1 6
4 0 1 9
7 0 1 11
7 4 1 11
7 2 0
```

Sample Output 2

```
6
```

Sample Input 3	Sample Output 3
2 9 2 1 1 6 2 1 0 0 2 1 4 0 3 1 2 3 3 1 3 3 1 2 4 5 3 0 1 8 5 1 1 6 5 3 1 7 7 3 1 8 7 0 0	2

Jaguar Jump

Problem ID: jaguarjump

Trekking in the Amazon rainforest sounds like a daunting task, but Jarvis and his friends are well-prepared and have found a good guide. Bringing the right equipment is of course essential: On top of the list is mosquito repellent, closely followed by waterproof covers to avoid a permanently wet rucksack.

The first couple of days were a pleasant experience for Jarvis, except for some annoying insects here and there. It also took some time to get used to all the noises during the night, and uh, pro tip: Close your mosquito nets before going to bed. But apart from a big snake that lead to a smaller detour, things have gone very smoothly.

. . . that is, until the jaguar appeared. Cutely named "El Investigador" by the local community, the jaguar is mostly curious and investigative when it comes to humans, and tends to keep a safe distance. Though after every night, "El Investigador" has been seen closer and closer to the camp we set up.

This morning, the tour guide is more unnerved than usual: He has spotted the jaguar very close to the camp! As we all know, Jaguars are known for their incredible jump distances, and so he's afraid "El Investigador" could just jump straight into the camp.

You, on the other hand, aren't quite so sure about that. But you're not confident either, so you decide to fetch some real-time satellite images and quickly write a program to check whether the jaguar can jump straight into the camp.

Input

The first line contains three integers, W, H and D, denoting the width and height of the satellite image, along with the maximal distance "El Investigador" can jump. Then follows a grid, spanning H lines, each with exactly W ASCII-characters each. The symbol "`@`" denotes the location of the camp, and the symbol "`J`" denotes the location of "El Investigador".

Output

Output "`no jumpscares here`" if the Euclidean distance between the camp and "El Investigador" is strictly greater than D, otherwise output "`the guide is right`". The distance between two grid locations is the distance between their (x, y)-coordinates.

Limits

- $1 < W, H \leq 50$
- $0 < D < 72$
- The symbols in the grid will only be one of "`.,:;-+*=~_^#%@J`"
- There is exactly one "`@`" and one "`J`" on the grid

Sample Input 1	Sample Output 1
10 5 6 ~~~~,...,+ ~~~~~~_+.+ ~~=^_~~_== ~~*#*_~~__ J_+#__@~~+	the guide is right

Sample Input 2	Sample Output 2
6 7 3 ;:#_%^ ~~_@_^ ~~~__^ =~~~~* ;,J~~~ ..,-~~ --.^=~	no jumpscares here