Problem A. "Aaawww..." or "Aaayyy!!!"

Source file name:	Aaawww.c, Aaawww.cpp, Aaawww.java, Aaawww.py
Input:	`Standard`
Output:	`Standard`

The audience going "Aaawww...". © VIA FotoCollectief on Flickr, used with permission

You are a hardcore supporter for one of the teams at the Benelux Algorithm Programming Contest (BAPC). Unfortunately, the staff found you entering the contest area and popping other teams' balloons. As a result, you were kicked out of the building and you will not be allowed back in until the contest is over. As a hardcore supporter, you would not want to miss the award ceremony and not know the final rank of your favorite team while everyone inside already knows! Luckily, you know a way to follow the award ceremony from outside the building.

One hour before the end of the contest, the scoreboard freezes. Since you can access the frozen scoreboard online, you know the rank of each team and for which problems they have accepted, rejected or pending submissions. During the award ceremony, the results of all pending submissions are revealed, starting with the leftmost pending submission of the lowest ranking team with a pending submission. After every reveal, the scoreboard is updated and the next pending submission is chosen in the same way.

The BAPC is known for its great audience participation during the award ceremony. When the result of a pending submission is about to be revealed, the audience chants "Ooohhh..." in anticipation. When the submission is rejected, the audience utters a sad "Aaawww...". When the submission is accepted, the crowd goes wild with an "Aaayyy!!!". If an accepted submission causes a team to rise in the ranking, this chant goes on for longer. Specifically, an extra 'y' is added for each team they pass. For example, if they pass five teams, the audience will yell "Aaayyyyyyyy!!!" (with eight 'y's). While you cannot make out any words of the announcer, you can clearly hear the enthusiastic audience from outside. You want to use this to determine the final rank of your favorite team.

Input

The input consists of:

- One line with three integers n, m, and r ($2 \leq n \leq 100$, $1 \leq m \leq 100$, $1 \leq r \leq n$), the number of teams, the number of problems, and the rank of your favorite team in the frozen scoreboard.
- n lines with m characters, each character being either '`A`', '`R`', '`P`', or '`N`', indicating that a team's submission is either accepted, rejected, pending, or nothing. The teams are sorted by their rank in the frozen scoreboard, i.e. descending by number of accepted submissions in the frozen scoreboard, and further tie-breaking rules guarantee that all teams have a distinct rank.
- For each pending submission, one line with two strings, containing the audience chant at the reveal of the pending submission, in chronological order. The first string is "`Ooohhh...`" and the second string is either "`Aaawww...`" or "`Aaayyy!!!`", with an additional '`y`' for each team that is passed when rising in the ranking.

Output

Output the final rank of your favourite team.

Example

Input	Output
2 3 2 AAP APR Ooohhh... Aaayyyy!!! Ooohhh... Aaawww...	1
2 3 2 AAP APR Ooohhh... Aaayyyy!!! Ooohhh... Aaayyyy!!!	2
4 4 3 AAPP PNAA PPAA NAPN Ooohhh... Aaayyyy!!! Ooohhh... Aaayyyyyy!!! Ooohhh... Aaawww... Ooohhh... Aaayyy!!! Ooohhh... Aaayyyy!!! Ooohhh... Aaayyyy!!!	1

Problem B. Buggy Blinkers

Source file name:	Buggy.c, Buggy.cpp, Buggy.java, Buggy.py
Input:	`Standard`
Output:	`Standard`

Recently, your car underwent a software update. Now, if you use the blinkers too much, the car shuts down, reporting a "buffer overflow", whatever that means! On the bright side, you are now welcome at the Broken-down Automobile Preservation Convention (BAPC).

A car indicating to turn left.
CC BY-SA 3.0 by Scheinwerfermann on Wikimedia Commons

You found out late, so you have to drive there as quickly as possible! Still, of course, you have to obey all traffic rules. At each intersection, you should follow these rules, regardless of whether an intersection has roads in all directions or not:

- When turning left (or right) at an intersection, the left (or right) blinker must be on.
- When driving straight ahead, the blinkers must be off.
- U-turns are not allowed, i.e. you are not allowed to turn back the way you came.

To play it safe with your blinkers, you decide you are going to activate them at most k times. Luckily, you can still deactivate them at any time. This seems rather limiting, but you make one shrewd observation: as long as you keep your blinkers on (they do not turn off automatically), you can keep turning in the same direction.

The road network consists of intersections with roads between them. Roads always start and end in one of the four cardinal directions: north, east, south, or west. Furthermore, they never start and end at the same intersection. As an example, consider sample cases 1 through 3, visualized in Figure 1 (next page). These samples only differ in their value of k.

To simplify navigation, you assume that each road can be traversed in the same amount of time, i.e. each road is considered to be of length 1. Find the shortest route from your current location to the BAPC, ensuring that you do not activate the blinkers more than k times. From your current location, you can drive in any direction without using your blinkers.

Input

The input consists of:

- One line with two integers n and k ($1 \leq n \leq 5000$, $0 \leq k \leq 20$), the number of intersections and the number of times the blinkers can be activated.
- n lines, the ith of which contains four integers v_i^{N}, v_i^{E}, v_i^{S}, and v_i^{W} ($0 \leq v_i^{\mathrm{N}}, v_i^{\mathrm{E}}, v_i^{\mathrm{S}}, v_i^{\mathrm{W}} \leq n$), the intersections that can be reached by taking the north, east, south, and west road from intersection i, or 0 to indicate that the road does not exist.

You start at intersection 1, and the BAPC is located at intersection n. Each intersection i has at most one road to each other intersection j. If this road exists, then intersection j has exactly one road to intersection i as well.

Output

If it is possible to drive from intersection 1 to n using the blinkers at most k times, output the length of the shortest such route. Otherwise, output "`impossible`".

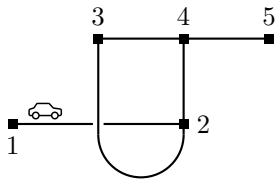


Figure 1. Visualization of the first, second, and third example input.

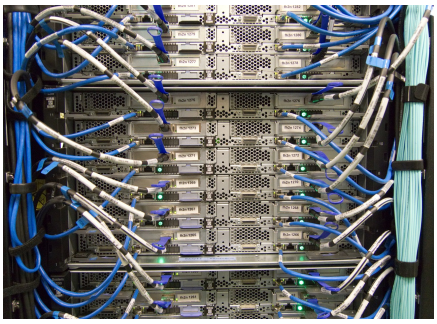
Example

Input	Output
5 2 0 2 0 0 4 0 3 1 0 4 2 0 0 5 2 3 0 0 0 4	3
5 1 0 2 0 0 4 0 3 1 0 4 2 0 0 5 2 3 0 0 0 4	4
5 0 0 2 0 0 4 0 3 1 0 4 2 0 0 5 2 3 0 0 0 4	impossible
5 2 0 2 0 0 4 0 3 1 0 4 2 0 0 0 2 3 0 0 0 0	impossible

Problem C. Chaotic Cables

Source file name:	Cables.c, Cables.cpp, Cables.java, Cables.py
Input:	`Standard`
Output:	`Standard`

Your friend Claas is in charge of designing the network for the newly constructed computer lab. Aware of the critical importance of efficiency in network design, Claas opted for the sophisticated Binary Access Point Configuration (BAPC) network topology.



Visualization of a possible BAPC network. CC0 by David Lohner on Flickr

A network is classified as a BAPC network precisely if we can assign a binary address of a fixed length to each device within the network, ensuring that:

1. Devices are connected if and only if their addresses differ in exactly one bit.
2. Each possible address is assigned to exactly one device.

Claas started out wiring devices together, but as the intricate web of connections began to take shape, doubt crept into his mind. Was the network he painstakingly constructed truly a BAPC network?

Help Claas determine if the network is a BAPC network.

Input

The input consists of:

- One line with two integers n and m ($2 \leq n \leq 2 \cdot 10^5$, $1 \leq m \leq 2 \cdot 10^5$) the number of devices and the number of wires in the network.
- m lines with integers a and b ($1 \leq a, b \leq n$, $a \neq b$), indicating that there is a wire between devices a and b.

It is guaranteed that each pair of devices is connected by at most one wire.

Output

Output "`yes`" if the network is a BAPC network. Otherwise, output "`no`".

Example

Input	Output
4 3 1 2 2 3 1 4	no
8 12 1 2 6 2 8 2 3 1 1 7 3 6 6 5 3 4 8 7 8 5 7 4 5 4	yes

Problem D. Dialling Digits

Source file name:	Dialling.c, Dialling.cpp, Dialling.java, Dialling.py
Input:	`Standard`
Output:	`Standard`

CC BY 2.0 by Elliot Brown on Flickr, modified

Disaster struck at the Billboards And Phone-numbers Company! Due to a bug in their database system, they lost the corresponding mnemonic phrases for each of the registered phone numbers. These mnemonic phrases are typically used on billboards, to make a phone number for an advertisement easier to remember by people who read them. To dial the phone number from a mnemonic phrase, you simply have to press the number keys corresponding to each letter, as shown in Figure 2. For example, the phone number "`1-800-BAPC`" would be dialled as "`1-800-2272`".

Using this information and a given list of valid words, reconstruct the possible mnemonic phrases for each registered phone number. Each mnemonic phrase consists of exactly one word from the word list. In the input, you will only receive the part of the phone number that should be exactly linked to possible mnemonic phrases, so it does not include the "`1-800-`" prefix (or any other prefix).

Figure 2. The keypad of a telephone, where some numbers are assigned several letters.

Public Domain by Sakurambo on Wikimedia Commons, modified

Input

The input consists of:

- One line with two integers n and m ($1 \leq n, m \leq 10^5$, $n \cdot m \leq 10^5$), the number of words and the number of phone numbers.
- n lines, each with a word w ($1 \leq |w| \leq 10$), consisting only of English lowercase letters (`a-z`). The words are unique and given in alphabetical order.
- m lines, each with a phone number p ($1 \leq |p| \leq 10$), consisting only of digits that correspond to letters in a mnemonic phrase (`2-9`).

Output

For each phone number, output the number of words that are a valid mnemonic phrase for this phone

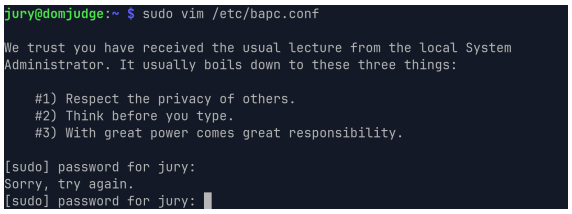
number, followed by these words in alphabetical order.

Example

Input	Output
5 3 algorithm bapc benelux contest progaming 2272 424242 2363589	1 bapc 0 1 benelux
3 1 peer reds refs 7337	3 peer reds refs
7 3 black judge my of quartz sphinx vow 25225 782789 774466	1 black 1 quartz 0
5 1 and bland e land of 63	1 of

Problem E. Expected Error

Source file name:	Error.c, Error.cpp, Error.java, Error.py
Input:	`Standard`
Output:	`Standard`



You are typing your password at a `sudo` prompt, but suddenly, one of your fingers slipped onto a wrong key. Because the terminal hides the characters that you type, you are uncertain whether you have typed an extra character. To finish typing your password, you consider three possible strategies.

1. *Continue* to type the rest of your password.
2. Press *backspace* to delete the last character and type the rest of your password.
3. *Restart* typing your password from scratch.

To determine the optimal strategy, make use of the following typing speed assumptions.

- Typing any character of your password takes 0.1 seconds.
- Pressing backspace or submitting your password also takes 0.1 seconds.
- To restart typing your password, you delete all characters, which takes 0.3 seconds.
- If you submit the wrong password, it takes an additional 0.3 seconds to realize and start typing your password at a new, empty prompt.

You are given the number of characters in your password n, the number of correctly typed characters k before your finger slipped, and a probability of $p\%$ indicating the likelihood that you pressed a wrong key and ended up with $k+1$ characters. Assuming you make no further errors, determine which strategy yields the lowest expected time to finish typing your password.

Input

The input consists of:

- One line with three integers n, k, and p ($1 \leq n \leq 1000$, $1 \leq k \leq n$, $0 \leq p \leq 100$), the number of characters in your password, the number of characters you correctly typed before your finger slipped, and the probability percentage that you pressed a wrong key.

Output

Output one of the strings "`continue`", "`backspace`", or "`restart`" indicating the optimal strategy. It is guaranteed that the expected time of the optimal strategy is at least one millisecond shorter than the other strategies.

Example

Input	Output
10 8 20	continue
10 8 80	backspace
10 2 50	restart
10 4 55	restart

Problem F. Fractal Area

Source file name:	Fractal.c, Fractal.cpp, Fractal.java, Fractal.py
Input:	`Standard`
Output:	`Standard`

The director of the local Mathematical Institute has decided to brighten up the walls by adding some pictures of Bounded Autosimilar Periodic Curves, which are geometric structures usually known as fractals. The director has some great ideas for beautiful fractals, but they are not sure whether these will fit on the walls of the institute.

Since these fractals will be painted all over the walls of the institute, the director has asked you to determine the area of these fractals, so they know exactly how much paint they will need to use for this.

The fractals are constructed from a polyline[1] between $(0, 0)$ and $(1, 0)$. Starting with an equilateral triangle with side length 1, each segment of the boundary is recursively replaced by a scaled and rotated version of the original polyline, so that the endpoints and orientation match.

The fractal from the second example.

As an example, consider the first sample case, visualized in Figure 3. The resulting fractal in this case is called the *Koch Snowflake*.

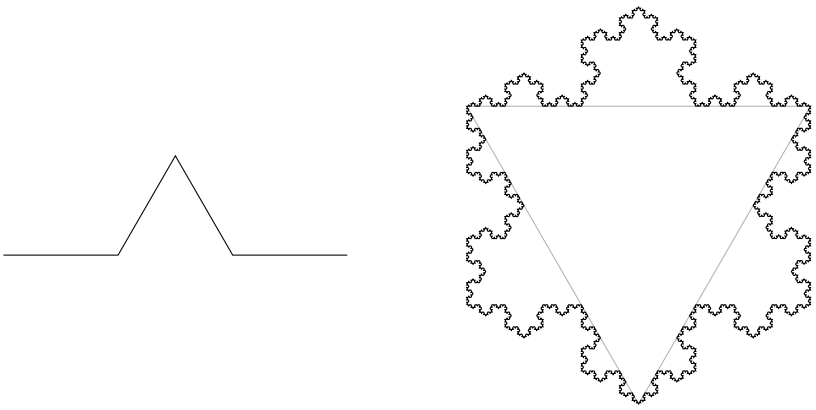


Figure 3. Visualization of the first example case, with the given polyline on the left and the resulting fractal on the right.

Input

The input consists of:

- One line with an integer n ($2 \leq n \leq 1000$), the number of points defining the polyline.
- n lines with two floating point numbers x and y ($0 \leq x \leq 1$, $|y| < 0.5$), the coordinates of a point defining the polyline.

All floating point numbers consist of exactly 6 digits behind the decimal point. It is guaranteed that the first point is $(0, 0)$ and the last point is $(1, 0)$. The resulting fractal converges and does not overlap with itself.

[1] A polyline is a shape made by connecting a series of straight line segments at their endpoints.

Output

Output the area of the resulting fractal.

Your answer should have an absolute or relative error of at most 10^{-6}.

Example

Input	Output
5 0.000000 0.000000 0.333333 0.000000 0.500000 0.288675 0.666667 0.000000 1.000000 0.000000	0.692820545
5 0.000000 0.000000 0.200000 0.000000 0.600000 -0.200000 0.500000 0.000000 1.000000 0.000000	0.237360528

Problem G. Grocery Greed

Source file name:	Grocery.c, Grocery.cpp, Grocery.java, Grocery.py
Input:	`Standard`
Output:	`Standard`

Paying by card for *only* the milk. CC PDM 1.0 by U.S. Department of Agriculture on Flickr

Recently, you have acquired the newest book in the self-help category: "Becoming A Professional Consumer", containing a wide variety of tips on how to buy as much as possible, while paying as little as possible. One of the things that you already discovered while reading the book is that you have been paying too much for your groceries all your life!

This works as follows: in a supermarket, you can decide to pay with card or with cash. If you pay with cash, the amount you have to pay gets rounded to the nearest multiple of €0.05, and if you pay with card, it does not. So, depending on your groceries, it can be cheaper if you pay with the right method! You can minimize your spendings even further by splitting your groceries into multiple groups, and paying separately for every group.

You have already decided on a list of the things that you are going to buy, and you know their prices. What is the cheapest way to buy all these groceries?

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 2 \cdot 10^5$), the number of items you want to buy.
- One line with n floating point numbers p ($0.05 \leq p \leq 100.00$), the prices of the items in euros. Each price is given in decimal form[1] with exactly two decimal places.

Output

Output the minimal total amount of money you need to buy all the groceries, in euros. Your answer should have exactly two decimal places.

Example

Input	Output
3 0.59 5.21 3.10	8.89
5 20.43 1.11 6.47 19.99 3.75	51.70
2 0.05 0.14	0.19
4 1.00 3.00 5.00 2.00	11.00
3 68.79 61.18 0.58	130.53

[1] When a floating-point number is written in decimal form, it is not in scientific notation.

Problem H. Human Pyramid

Source file name:	Human.c, Human.cpp, Human.java, Human.py
Input:	`Standard`
Output:	`Standard`

An example of a non-world record breaking pyramid.
CC BY-SA 3.0 by Amotoki on Wikimedia Commons

As chairman of the Building A Pyramid Committee, you are specialized in breaking one specific world record: building the highest human pyramid. Unfortunately, you only know a limited number of people who are willing to be in the pyramid. After all, building human pyramids does not make much money, so most people are volunteers.

A full human pyramid of height h consists of h layers of people. As seen from below, it has h people on the first layer, $h-1$ on the second, $h-2$ on the third, and so forth until eventually the final layer has just a single person. To determine whether you can break the world record, you need to know how high a pyramid you can build. Given how many people are available, how tall is the highest possible human pyramid that these people can make?

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 10^{12}$), the number of people available to build the pyramid.

Output

Output the height of the highest possible pyramid you can build with n people.

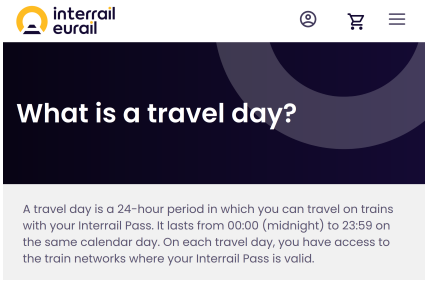
Example

Input	Output
3	2
12	4

Problem I. Interrail Pass

Source file name:	Interrail.c, Interrail.cpp, Interrail.java, Interrail.py
Input:	`Standard`
Output:	`Standard`

Interrail passes are the fun and cheap way to see more of Europe, especially if you combine your train trip with Businesslike And Penny-saving Computation! In particular, you would like to find the cheapest way to pay for your planned travels. You plan to take the train on n travel days, that are not necessarily consecutive. The individual fare is different for every day, and perhaps you can save money by buying some interrail passes.



Definition of a travel day from interrail.eu
© Eurail B.V., used non-commercially

There are k different types of interrail passes with varying costs. Each type of interrail pass can be obtained multiple times. An interrail pass is active for a period of p consecutive days, that starts on a day of your choice. The interrail pass covers the first d travel days during this period, which do not have to be consecutive. Note that an active interrail pass cannot be "paused": a day of travel counts towards the day count of each active pass, even when you pay the individual fare that day.

As an example, consider the fourth sample input, visualized in Figure 4. It is definitely cheaper to buy interrail passes than to pay 4 individual fares. The cheapest solution is to buy two interrail passes of the first type, rather than one interrail pass of the second type.

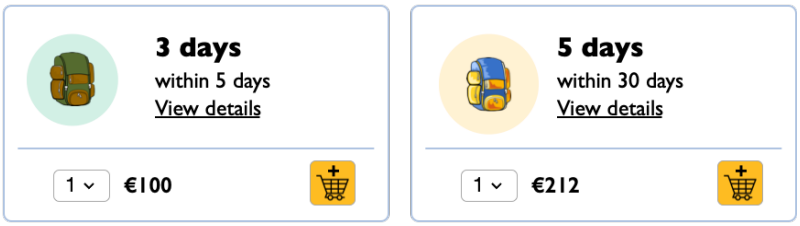


Figure 4. Visualization of the types of interrail passes for the fourth sample input in a webshop. The first one can be activated for a period of 5 days, and can be used for 3 days within that period. The second one has a period of 30 days, and can be used for 5 days during that period.

Input

The input consists of:

- One line with two integers n and k ($1 \leq n \leq 10\,000$, $0 \leq k \leq 100$), the number of planned travel days, and the number of types of interrail passes available.
- n lines, each with two integers t and f ($0 \leq t \leq 10^6$, $1 \leq f \leq 10^5$), the travel day and the individual fare for that day. The n travel days are distinct and given in increasing order.
- k lines, each with three integers p, d, and c ($1 \leq p \leq 10^6$, $1 \leq d \leq p$, $1 \leq c \leq 10^5$), indicating a type of interrail pass that is valid for a period of p days, covers the first d travel days in that period, and costs c.

Output

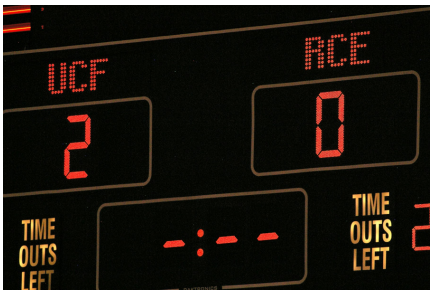
Output the minimum amount you need to spend to cover all your planned travels.

Example

Input	Output
2 1 0 10 1 10 2 2 15	15
2 1 0 10 2 10 2 2 15	20
3 1 0 10 1 10 2 10 5 2 15	25
4 2 3 80 5 90 24 70 26 60 5 3 100 30 5 212	200
4 1 42 9 43 2 44 9 45 9 4 3 20	29

Problem J. Jumbled Scoreboards

Source file name:	Jumbled.c, Jumbled.cpp, Jumbled.java, Jumbled.py
Input:	`Standard`
Output:	`Standard`



One of the pictures that you received for the first example input. CC BY 2.0 by Adam Baker on Flickr, modified

You were so hyped to attend the final game of the Ball And Paddle Competition, where the two best teams in the world compete to paddle as many balls into the opponent's goal as possible. But alas, you fell ill, and cannot join your friends. Luckily, your friends took lots of pictures during the match, and after the match concluded, they sent you all the pictures that they have. Because the messaging app uploads and downloads the pictures in parallel, you are wondering whether you received them in chronological order. It looks like the scoreboards in each picture are unique, and knowing that the score of a team can only increase over time, you should be able to figure this out. Feeling too sick to check the order of the pictures manually, you decide to write a program that checks temporal consistency based on the scoreboards that are in the picture.

Given a list of intermediate scores from the match, determine whether the scores are in chronological order.

Input

The input consists of:

- One line with an integer n ($1 \leq n \leq 100$), the number of pictures you received.
- n lines, each with two integers a and b ($0 \leq a, b \leq 100$), the scores of the two teams in one of the pictures.

Every pair of scores (a, b) in the input is unique.

The order of the scores in the input is the order in which you received the pictures.

Output

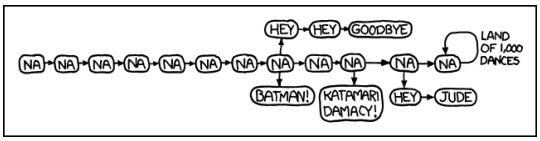
Output "`yes`" if the scores are in chronological order, or "`no`" if they are not.

Example

Input	Output
4 1 0 2 0 4 0 4 1	yes
3 0 0 1 0 0 2	no
5 1 2 0 0 4 3 2 3 5 5	no

Problem K. Karaoke Compression

Source file name:	Karaoke.c, Karaoke.cpp, Karaoke.java, Karaoke.py
Input:	`Standard`
Output:	`Standard`



Flowcharts are another compression method that you have considered. CC BY-NC 2.5 by Randall Munroe on xkcd.com

Next week, you will be hosting the Biannual Acoustic Pop-song Convention. Of course, this convention also needs to include a karaoke night, featuring all your favourite acoustic pop songs! To impress all attendees, you have decided to prepare by learning the lyrics of all the songs by heart. But there is a problem: these lyrics are very long, so you will not have enough time left to learn all of this! However, you have noticed that a lot of the songs contain quite some repetitions. This gives you the idea of first compressing the lyrics, and then only learning the compressed version.

The compression scheme you will use works as follows. Let s be the string to compress. You select exactly one nonempty substring t of the lyrics, and replace as many occurrences of t in s as possible by a new character that did not occur in s before. Call the result of this s'. Now you only need to remember the substring t and the compressed string s'. You would like to know the minimal total length of these two strings, if you compress the lyrics in this manner.

As an example, consider the first sample case. In this case, you want to compress the lyrics "`nanananananananabatman`". If you replace the substring "`na`" by the character "`X`", the compressed string becomes "`XXXXXXXXbatman`". The total length of the substring and compressed string in this case is $2 + 14 = 16$. However, if you instead choose to replace the substring "`nana`", then the compressed string is "`XXXXbatman`" and the total length is $4 + 10 = 14$, which is optimal.

Input

The input consists of:

- One line with a string s ($1 \leq |s| \leq 5000$), the lyrics to compress.
  The string only consists of English lowercase letters (`a-z`).

Output

Output the minimal total length of the replaced substring and the compressed string.

Example

Input	Output
nanananananananabatman	14
abcabd	6
nocompression	14

Problem L. Lawful Limits

Source file name:	Lawful.c, Lawful.cpp, Lawful.java, Lawful.py
Input:	`Standard`
Output:	`Standard`

One late afternoon you are driving to get home in your Big And Pricey Car. You have had a long day and are eager to get home as soon as possible. Your country's road network has many roads with varying speed limits, and has one strange quirk: at some time t, the maximum speed on each road is raised. Because you want to get home as soon as possible, you instantly increase your speed to the new maximum speed of the road you are on at time t.

You start driving at time 0 at junction 1 and are going to n. What is the earliest time you can reach your destination? As an example, consider the first example case, visualized in Figure 5.

In this road network, the speed limit increases at 19:00.

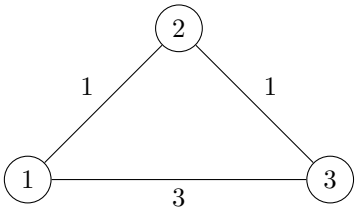


Figure 5. Visualization of the first example input. The edges are marked with their lengths. On all roads, the maximum speed is 1 before time t and 2 from time t onwards.

Input

The input consists of:

- One line with three integers n, m, and t ($2 \leq n \leq 10^5$, $1 \leq m \leq 10^5$, $0 \leq t \leq 10^9$), the number of junctions, the number of roads, and the time the speed limit increases.
- m lines, each with five integers x, y, ℓ, v, and w ($1 \leq x, y \leq n$, $1 \leq \ell \leq 10^9$, $1 \leq v < w \leq 10^9$), the start and end junction of a road, length of this road, and the speed limits on this road before time t and from time t onwards.

There is at most one road between any two junctions, and one can travel in both directions on any road. No road leads from one junction to that same junction. It is guaranteed that there is always a path between any two junctions.

Output

Output the minimum amount of time it takes to get from the start to your destination.

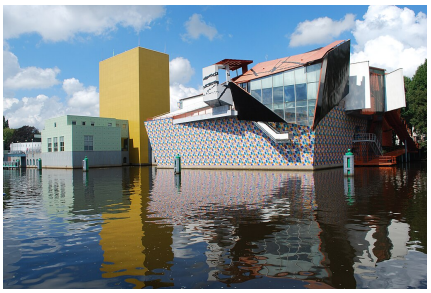
Your answer should have an absolute or relative error of at most 10^{-6}.

Example

Input	Output
3 3 1 1 2 1 1 2 2 3 1 1 2 1 3 3 1 2	1.5
2 1 1 1 2 3 1 2	2.0
4 4 6 1 2 30 4 6 1 3 12 6 8 2 4 16 4 8 3 4 30 5 10	7.0

Problem M. Museum Visit

Source file name:	Museum.c, Museum.cpp, Museum.java, Museum.py
Input:	`Standard`
Output:	`Standard`



The Groninger Museum, as seen from the canals. CC BY-SA 4.0 by Rob Koster on Wikimedia Commons

Every day is different in the *Groninger Museum*. Some days are nice, peaceful and quiet, and you can spend all day looking at the beautiful paintings, sculptures, and other artworks. Other days are busier, when weekends or public holidays fill the museum with hasty visitors, increased prices and screaming children. This discomfort varies a lot: some busy days are better because of extra *studentenkorting* (student discount) and some of the quiet days get worse because of earthquake risks.

The museum also regularly hosts special limited-time exhibitions, such as those on the local football club FC Groningen, the Martinitoren, or the eierbal (a local delicacy). These exhibitions can be very irregular: some last for weeks, some last only a day, and there may be multiple exhibitions on the same day.

As a proud *Grunneger*, you want to visit each exhibition at least once. Luckily, you are subscribed to the newsletter so you know the start and end days of all the exhibitions in advance. Additionally, since you are a regular visitor at the museum, you have observed all the crowd and earthquake patterns, so you know exactly how much discomfort you will receive when you visit the museum on any specific day.

On which days should you visit the museum in order to minimize your total discomfort while still seeing all exhibitions that are planned in the foreseeable future? As an example, consider the first sample case. To minimize your total discomfort, you should visit the first two exhibitions on the second day and the last exhibition on the fourth or fifth day.

Input

The input consists of:

- One line with two integers n and m ($1 \leq n, m \leq 2 \cdot 10^5$), the number of days in the foreseeable future and the number of exhibitions planned in those days.
- One line with n integers c ($1 \leq c \leq 10^9$), describing for each day the discomfort you will receive when you visit the museum.
- m lines, each with two integers s and e ($1 \leq s \leq e \leq n$), describing the start and end day of an exhibition. The start and end days are inclusive: the exhibition can be visited on day s, day e, and any day in between.

Output

Output the minimum total discomfort you will receive when visiting all exhibitions of the Groninger Museum.

Example

Input	Output
5 3 1 1 3 1 1 1 3 2 3 3 5	2
6 3 1 2 4 4 2 1 1 4 2 5 3 6	3
11 2 3 1 4 1 5 9 2 6 5 3 5 5 10 1 1	5