Problem A
Birthday Cake

Photo by 4rank

On his birthday, John's parents made him a huge birthday cake! Everyone had a wonderful dinner, and now it's time to eat the cake. There are n candles on the cake. John wants to divide the cake into n pieces so that each piece has exactly one candle on it, and there are no leftover pieces. For that, he made m cuts across the cake. Could you help check if John's cuts successfully divide the candles on the cake?

Formally, the cake is a circle of radius r centered at $(0, 0)$. The candles are n distinct points located strictly inside the circle. Each cut is a straight line $ax + by + c = 0$, described by three coefficients a, b, and c.

Input

Input starts with three integers n ($1 \le n \le 50$), m ($1 \le m \le 15$), and r ($1 \le r \le 100$) on the first line.

The next n lines give the locations of the candles. Each line has two integers x and y giving the coordinates of one candle ($0 \le \sqrt{x^2 + y^2} < r$).

The next m lines give the coefficients of the cutting lines. Each line has three integers a, b, and c ($0 \le |a|, |b| \le 100, 0 \le |c| \le 20\,000$) describing a line of the form $ax + by + c = 0$. The values a and b are not both zero.

All candles and lines are distinct. No candle is on a cut line. No line is completely outside or tangent to the cake. The input guarantees that the number of cake pieces remains the same if any cut line is shifted by at most 10^{-4} in any direction. The input also guarantees that each candle remains in the interior of the same piece of cake if its position is shifted by at most 10^{-4} in any direction.

Output

Output "`yes`" if John's cuts successfully divide the cake so that each piece he obtains has exactly one candle on it. Otherwise, output "`no`".

Sample Input 1	Sample Output 1
4 2 3 0 1 1 0 -1 0 0 -1 -1 1 0 2 1 0	yes

Sample Input 2	Sample Output 2
4 3 3 0 1 1 2 -1 2 0 -1 -1 1 -2 -1 -1 2 0 -1 0	no

Sample Input 3	Sample Output 3
3 2 3 2 1 0 0 -1 -2 1 1 -2 3 6 12	yes

Sample Input 4	Sample Output 4
3 1 2 0 0 -1 1 1 -1 -2 2 1	no

Problem B
Bumped!

Photo by jtosh/Flickr

Peter returned from the recently held ACM ICPC World Finals only to find that his return flight was overbooked and he was bumped from the flight! Well, at least he wasn't beat up by the airline and he's received a voucher for one free flight between any two destinations he wishes.

He is already planning next year's trip. He plans to travel by car where necessary, but he may be using his free flight ticket for one leg of the trip. He asked for your help in his planning.

He can provide you a network of cities connected by roads, the amount it costs to buy gas for traveling between pairs of cities, and a list of available flights between some of those cities. Help Peter by finding the minimum amount of money he needs to spend to get from his hometown to next year's destination!

Input

The input consists of a single test case. The first line lists five space-separated integers n, m, f, s, and t, denoting the number of cities n ($0 < n \leq 50\,000$), the number of roads m ($0 \leq m \leq 150\,000$), the number of flights f ($0 \leq f \leq 1\,000$), the number s ($0 \leq s < n$) of the city in which Peter's trip starts, and the number t ($0 \leq t < n$) of the city Peter is trying to travel to. (Cities are numbered from 0 to $n - 1$.)

The first line is followed by m lines, each describing one road. A road description contains three space-separated integers i, j, and c ($0 \leq i, j < n, i \neq j$ and $0 < c \leq 50\,000$), indicating there is a road connecting cities i and j that costs c cents to travel. Roads can be used in either direction for the same cost. All road descriptions are unique.

Each of the following f lines contains a description of an available flight, which consists of two space-separated integers u and v ($0 \leq u, v < n$, $u \neq v$) denoting that a flight from city u to city v is available (though not from v to u unless listed elsewhere). All flight descriptions are unique.

Output

Output the minimum number of cents Peter needs to spend to get from his home town to the competition, using at most one flight. You may assume that there is a route on which Peter can reach his destination.

Sample Input 1

```
8 11 1 0 5
0 1 10
0 2 10
1 2 10
2 6 40
6 7 10
5 6 10
3 5 15
3 6 40
3 4 20
1 4 20
1 3 20
4 7
```

Sample Output 1

```
45
```

Sample Input 2

```
8 11 1 0 5
0 1 10
0 2 10
1 2 10
2 6 40
6 7 10
5 6 10
3 5 15
3 6 40
3 4 20
1 4 20
1 3 30
4 7
```

Sample Output 2

```
50
```

Problem C
Canonical Coin Systems

Image by Diane Wiens, Used with permission

A *coin system* S is a finite (nonempty) set of distinct positive integers corresponding to coin values, also called *denominations*, in a real or imagined monetary system. For example, the coin system in common use in Canada is $\{1, 5, 10, 25, 100, 200\}$, where 1 corresponds to a 1-cent coin and 200 corresponds to a 200-cent (2-dollar) coin. For any coin system S, we assume that there is an unlimited supply of coins of each denomination, and we also assume that S contains 1, since this guarantees that any positive integer can be written as a sum of (possibly repeated) values in S.

Cashiers all over the world face (and solve) the following problem: For a given coin system and a positive integer amount owed to a customer, what is the smallest number of coins required to dispense exactly that amount? For example, suppose a cashier in Canada owes a customer 83 cents. One possible solution is $25+25+10+10+10+1+1+1$, i.e., 8 coins, but this is not optimal, since the cashier could instead dispense $25 + 25 + 25 + 5 + 1 + 1 + 1$, i.e., 7 coins (which *is* optimal in this case). Fortunately, the Canadian coin system has the nice property that the *greedy algorithm* always yields an optimal solution, as do the coin systems used in most countries. The greedy algorithm involves repeatedly choosing a coin of the largest denomination that is less than or equal to the amount still owed, until the amount owed reaches zero. A coin system for which the greedy algorithm is always optimal is called *canonical*.

Your challenge is this: Given a coin system $S = \{c_1, c_2, \ldots, c_n\}$, determine whether S is canonical or non-canonical. Note that if S is non-canonical then there exists at least one *counterexample*, i.e., a positive integer x such that the minimum number of coins required to dispense exactly x is less than the number of coins used by the greedy algorithm. An example of a non-canonical coin system is $\{1, 3, 4\}$, for which 6 is a counterexample, since the greedy algorithm yields $4 + 1 + 1$ (3 coins), but an optimal solution is $3 + 3$ (2 coins). A useful fact (due to Dexter Kozen and Shmuel Zaks) is that if S is non-canonical, then the smallest counterexample is less than the sum of the two largest denominations.

Input

Input consists of a single case. The first line contains an integer n $(2 \leq n \leq 100)$, the number of denominations in the coin system. The next line contains the n denominations as space-separated integers $c_1\ c_2\ \ldots\ c_n$, where $c_1 = 1$ and $c_1 < c_2 < \ldots < c_n \leq 10^6$.

Output

Output "`canonical`" if the coin system is canonical, or "`non-canonical`" if the coin system is non-canonical.

Sample Input 1	Sample Output 1
4 1 2 4 8	canonical

Sample Input 2	Sample Output 2
3 1 5 8	non-canonical

Sample Input 3	Sample Output 3
6 1 5 10 25 100 200	canonical

Problem D
Cat and Mice

Photo by katie_mccolgan/Flickr

Everyone knows that cats love to eat mice. Naturally, when given the opportunity, any cat wants to eat as many mice as possible.

It just so happens that Cartesian Cat lives on the Cartesian Plane with her home located at $(0, 0)$. Obviously, none of the mice live at this location, they are smarter than that! However, the mice aren't terribly smart. They live on the plane with Cartesian Cat and at time $t = 0$, stick their heads above the ground, in sight of Cartesian Cat. Each of the mice stays in sight at its location for a specified amount of time before ducking back underground, where Cartesian Cat can't get it.

Cartesian Cat has a plan. At time $t = 0$, she'll run straight towards a mouse, eat it instantaneously, and then head for another mouse, repeating the sequence until all the mice are eaten. She encounters two problems however: each time she eats a mouse her velocity reduces (due to her increased mass) by a constant multiplicative factor, m. That is, if her velocity is v before eating a mouse, then after eating the mouse her velocity is $v \cdot m$. Also, if she doesn't reach a mouse before it ducks underground, she won't be able to eat it. But she can eat a mouse if she reaches it precisely at the time it ducks underground.

Since all cats are efficient by nature, help Cartesian Cat determine what her minimum initial velocity must be if she hopes to eat all of the mice. Assume that she eats the mice in an optimal order.

Input

The first line of input contains a single positive integer, n (where $1 \leq n \leq 15$), representing the number of mice. Each of the following n lines describes a mouse given as three space-separated integers x, y, and s. This indicates that a mouse is located at (x, y) and will duck underground at $t = s$ seconds. The values of x and y are in the range $[-1\,000, 1\,000]$ and s is in the range $[1, 10\,000]$. No two mice are at the same location. The last line of input contains a single floating-point number, m, specified to two decimal places, in the range $[0.75, 0.99]$, representing the multiplicative factor by which the cat slows down after eating a single mouse.

Output

Output the minimum initial velocity (in units per second) necessary for Cartesian Cat to allow her to eat all of the mice, given that she eats them in the optimal order. Your answer should be correct within a relative or absolute error of 10^{-3}.

Sample Input 1	Sample Output 1
1 3 4 2 .75	2.4999999987500003

Sample Input 2	Sample Output 2
2 0 100 10 0 -100 100 .80	9.999999999000002

Sample Input 3	Sample Output 3
2 0 100 10 0 -100 15 .80	23.33333333177778

Problem E
Company Picnic

Each year, your employer hosts a company picnic. This event features a three-legged race, a race where two runners work as a team, running side-by-side with the legs between them tied together. It is more difficult to run like that, so teams run at a speed that is the minimum of the running speed of the two team members. For example, if Mildred can run 4.4 meters per second and Ken can run 4.0 meters per second, then, as a team, they will run 4.0 meters per second.

To improve company morale, all teams are chosen so they include an employee and the supervisor they report directly to. In the organization chart below, Mildred could be on a team with Ken (running at 4.0 meters per second) or with Zack (running at 4.2 meters per second), but Mildred could not be on a team with Barbara.

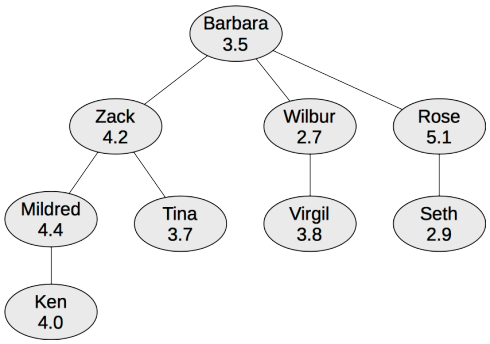

Figure E.1: Organization chart illustrating the sample input below.

Given a description of the company organizational chart, your job is to create as many teams as possible for the upcoming race, provided each employee can only be on one team. To make the race exciting, you want to choose the fastest teams possible, so, while forming as many teams as you can, you must pair up team members so that the average team running speed is maximized. For example, in the chart above, you could form four teams by pairing Mildred with Ken, Zack with Tina, Wilbur with Virgil and Rose with Seth. However, a better solution would pair Rose with Barbara instead of Seth. This would still give you four teams, but it would give you a greater average speed for the teams.

Input

The first line of input contains an integer n ($2 \leq n \leq 1\,000$) giving the total number of employees in the company. This is followed by n lines, each describing an employee. Each of these lines contains three space-separated values, first the name of the employee, then a real number giving their running speed in meters per second, and, finally, the name of their immediate supervisor in the organization chart. The organization chart is guaranteed to form a tree, with the CEO at the root. Since the CEO does not report to anyone, the input just gives "CEO" as their supervisor (no one's name is CEO). For all other employees, the supervisor is the name of another employee listed elsewhere in the input. Employee names are all unique and consist of 1 to 12 upper- and lower-case letters (a–z), and running speeds are all in the plausible range from 2.2 meters per second up to 5.3 meters per second. Running speeds have at most 3 digits after the decimal point.

Output

Print the largest number of three-legged-race teams that can be formed, followed by the maximum possible average running speed for that number of teams. The speed should be accurate to within 0.001 meters per second.

Sample Input 1	Sample Output 1
9 Barbara 3.5 CEO Rose 5.1 Barbara Seth 2.9 Rose Virgil 3.8 Wilbur Mildred 4.4 Zack Wilbur 2.7 Barbara Tina 3.7 Zack Ken 4.0 Mildred Zack 4.2 Barbara	4 3.47500000

Problem F
GlitchBot

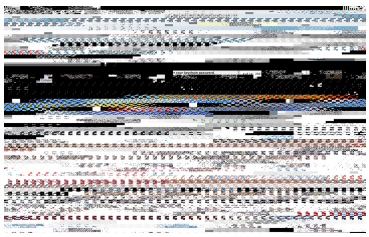

Image by Shawn Allen

One of our delivery robots is malfunctioning! The job of the robot is simple; it should follow a list of instructions in order to reach a target destination. The list of instructions is originally correct to get the robot to the target. However, something is going wrong as we upload the instructions into the robot's memory. During the upload, one random instruction from the list takes on a different value than intended. Yes, there is *always* a single bad instruction in the robot's memory and it *always* results in the robot arriving at an incorrect destination as it finishes executing the list of instructions.

The robot can execute the instructions "`Left`", "`Right`", and "`Forward`". The "`Left`" and "`Right`" instructions do not result in spatial movement but result in a 90-degree turn in the corresponding direction. "`Forward`" is the only instruction that results in spatial movement, causing the robot to move one unit in the direction it is facing. The robot always starts at the origin $(0, 0)$ of a grid and faces north along the positive y-axis.

Given the coordinates of the target destination and the list of instructions that the robot has in its memory, you are to identify a correction to the instructions to help the robot reach the proper destination.

Input

The first line of the input contains the x and y integer coordinates of the target destination, where $-50 \leq x \leq 50$ and $-50 \leq y \leq 50$. The following line contains an integer n representing the number of instructions in the list, where $1 \leq n \leq 50$. The remaining n lines each contain a single instruction. These instructions may be: "`Left`", "`Forward`", or "`Right`".

Output

Identify how to correct the robot's instructions by printing the line number (starting at 1) of an incorrect input instruction, followed by an instruction substitution that would make the robot reach the target destination. If there are multiple ways to fix the instructions, report the fix that occurs for the earliest line number in the sequence of instructions. There is always exactly one unique earliest fix.

Sample Input 1

```
3 2
11
Forward
Right
Forward
Forward
Left
Forward
Forward
Left
Forward
Right
Forward
```

Sample Output 1

```
8 Right
```

Sample Input 2

```
-1 1
3
Right
Left
Forward
```

Sample Output 2

```
1 Forward
```

Problem G
Greeting Card

Image by Varou.d

Quido plans to send a New Year greeting to his friend Hugo. He has recently acquired access to an advanced high-precision plotter and he is planning to print the greeting card on the plotter.

Here's how the plotter operates. In step one, the plotter plots an intricate pattern of n dots on the paper. In step two, the picture in the greeting emerges when the plotter connects by a straight segment each pair of dots that are exactly $2\,018$ length units apart.

The plotter uses a special holographic ink, which has a limited supply. Quido wants to know the number of all plotted segments in the picture to be sure that there is enough ink to complete the job.

Input

The first line of input contains a positive integer n specifying the number of plotted points. The following n lines each contain a pair of space-separated integer coordinates indicating one plotted point. Each coordinate is non-negative and less than 2^{31}. There are at most 10^5 points, all of them are distinct.

In this problem, all coordinates and distances are expressed in plotter length units, the length of the unit in the x-direction and in the y-direction is the same.

Output

The output contains a single integer equal to the number of pairs of points which are exactly $2\,018$ length units apart.

Sample Input 1	Sample Output 1
4 20180000 20180000 20180000 20182018 20182018 20180000 20182018 20182018	4

Sample Input 2	Sample Output 2
6 0 0 1680 1118 3360 0 5040 1118 6720 0 8400 1118	5

This page is intentionally left blank.

Problem H
Imperfect GPS

Photo by Aaron Parecki

Lots of runners use personal Global Positioning System (GPS) receivers to track how many miles they run. No GPS is perfect, though: it only records its position periodically rather than continuously, so it can miss parts of the true running path. For this problem we'll consider a GPS that works in the following way when tracking a run:

- At the beginning of the run, the GPS first records the runner's starting position at time 0.
- It then records the position every t units of time.
- It always records the position at the end of the run, even if the total running time is not a multiple of t.

The GPS assumes that the runner goes in a straight line between each consecutive pair of recorded positions. Because of this, a GPS can underestimate the total distance run.

For example, suppose someone runs in straight lines and at constant speed between the positions on the left side of Table 1. The time they reach each position is shown next to the position. They stopped running at time 11. If the GPS records a position every 2 units of time, its readings would be the records on the right side of Table 1.

Time	Position	Time	Position
0	$(0,0)$	0	$(0,0)$
3	$(0,3)$	2	$(0,2)$
5	$(-2,5)$	4	$(-1,4)$
7	$(0,7)$	6	$(-1,6)$
9	$(2,5)$	8	$(1,6)$
11	$(0,3)$	10	$(1,4)$
		11	$(0,3)$

Table 1: Actual Running Path on the left, GPS readings on the right.

The total distance run is approximately 14.313708 units, while the GPS measures the distance as approximately 11.650281 units. The difference between the actual and GPS distance is approximately 2.663427 units, or approximately 18.607525% of the total run distance.

Given a sequence of positions and times for a running path, as well as the GPS recording time interval t, calculate the percentage of the total run distance that is lost by the GPS. Your computations should assume that the runner goes at a constant speed in a straight line between consecutive positions.

Input

The input consists of a single test case. The first line contains two integers n ($2 \leq n \leq 100$) and t ($1 \leq t \leq 100$), where n is the total number of positions on the running path, and t is the recording time interval of the GPS (in seconds).

The next n lines contain three integers per line. The i-th line has three integers x_i, y_i ($-10^6 \leq x_i, y_i \leq 10^6$), and t_i ($0 \leq t_i \leq 10^6$), giving the coordinates of the i-th position on the running path and the time (in seconds) that position is reached. The values of t_i's are strictly increasing. The first and last positions are the start and end of the run. Thus, t_1 is always zero.

It is guaranteed that the total run distance is greater than zero.

Output

Output the percentage of the actual run distance that is lost by the GPS. The answer is considered correct if it is within 10^{-5} of the correct answer.

Sample Input 1	Sample Output 1
6 2 0 0 0 0 3 3 -2 5 5 0 7 7 2 5 9 0 3 11	18.60752550117103

Problem I
Odd Gnome

Photo by Chris Friese

According to the legend of Wizardry and Witchcraft, gnomes live in burrows underground, known as gnome holes. There they dig up and eat the roots of plants, creating little heaps of earth around gardens, causing considerable damage to them.

Mrs. W, very annoyed by the damage, has to regularly de-gnome her garden by throwing the gnomes over the fence. It is a lot of work to throw them one by one because there are so many. Fortunately, the species is so devoted to their kings that each group always follows its king no matter what. In other words, if she were to throw just the king over the fence, all the other gnomes in that group would leave.

So how does Mrs. W identify the king in a group of gnomes? She knows that gnomes travel in a certain order, and the king, being special, is always the only gnome who does not follow that order.

Here are some helpful tips about gnome groups:

- There is exactly one king in a group.
- Except for the king, gnomes arrange themselves in strictly increasing ID order.
- The king is always the only gnome out of that order.
- The king is never the first nor the last in the group, because kings like to hide themselves.

Help Mrs. W by finding all the kings!

Input

The input starts with an integer n, where $1 \leq n \leq 100$, representing the number of gnome groups. Each of the n following lines contains one group of gnomes, starting with an integer g, where $3 \leq g \leq 1\,000$, representing the number of gnomes in that group. Following on the same line are g space-separated integers, representing the gnome ordering. Within each group all the integers (including the king) are unique and in the range $[0, 10\,000]$. Excluding the king, each integer is exactly one more than the integer preceding it.

Output

For each group, output the king's position in the group (where the first gnome in line is number one).

Sample Input 1	Sample Output 1
3 7 1 2 3 4 8 5 6 5 3 4 5 2 6 4 10 20 11 12	5 4 2

Problem J
Progressive Scramble

Photo by Chilanga Cement

You are a member of a naive spy agency. For secure communication, members of the agency use a very simple encryption algorithm – which changes each symbol in the message 'progressively', i.e., based on the symbols preceding it. The allowed symbols are space and the 26 lowercase English letters. For encryption purposes we assign them the values 0 (for space) and 1 through 26 (for a–z). We'll let $v(s)$ represent the numeric value of symbol s.

Consider a message with symbols $s_1, s_2, \ldots, s_n$. The encryption algorithm starts by converting the first symbol s_1 into its associated value $u_1 = v(s_1)$. Then for each subsequent symbol s_i in the message, the computed value is $u_i = v(s_i) + u_{i-1}$ — the sum of its associated value and the computed value for the previous symbol. (Note that when there is a space in the input message, the previous scrambled letter is repeated.) This process continues until all the u_i are computed.

At this point, the message is a sequence of numeric values. We need to convert it back to symbols to print it out. We do this by taking the value u_i modulo 27 (since there are 27 valid symbols), and replacing that value with its corresponding symbol. For example, if $u_i = 32$, then $32 \bmod 27 = 5$, which is the symbol 'e' (since $v(e) = 5$).

Let's look at an example. Suppose we want to encrypt the string "my pie".

1. First, convert each symbol s_i into $v(s_i)$: $[13, 25, 0, 16, 9, 5]$.
2. Next, compute each u_i: $[13, 38, 38, 54, 63, 68]$.
3. Then, use modulus on the u_i: $[13, 11, 11, 0, 9, 14]$.
4. Finally, convert these back to symbols: "mkk in".

Create a program that takes text and encrypts it using this algorithm, and also decrypts text that has been encrypted with this algorithm.

Input

The input to your program consists of a single integer $1 \leq n \leq 100$ on its own line. This number is followed by n lines, each containing the letter 'e' or 'd', a single space, and then a message made up of lowercase letters (a–z) and spaces, continuing to the end of the line. Each message is between 1 and 80 characters long. The letters 'd' and 'e' indicate that your program decrypts or encrypts the subsequent string, respectively.

Output

Output the result of encrypting or decrypting each message from the input on its own separate line. Note that differences in whitespace are significant in this problem. Therefore your output must match the correct output character-for-character, including spaces.

Sample Input 1

```
7
e testing multiple letters rrrrrrrrrrrrr
e this particularly long  sentence can test encryption
d tajbbrsjcloiuvmywwhwjqqqinauzmpuuxyllejbvv nqhfvoxlz
e my pie
d mkk in
e the quick brown fox jumps over the lazy dog
d taffwqzbmmofuqddjyvvezlatthchzzs eeqrqoosgn
```

Sample Output 1

```
tyqjsfmmzteygwhmmycwpulddvmdvmdvmdvmdv
tajbbrsjcloiuvmywwhwjqqqinauzmpuuxyllejbvv nqhfvoxlz
this particularly long  sentence can test encryption
mkk in
my pie
taffwqzbmmofuqddjyvvezlatthchzzs eeqrqoosgn
the quick brown fox jumps over the lazy dog
```

Problem K
Space Probe

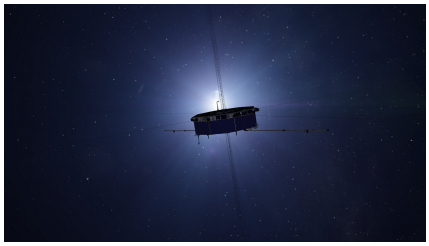

Photo by NASA Goddard Space Flight Center

The space probe is out of control! It's going to start its measurement sequence sometime between two given times t_1 and t_2 (measured in seconds), but we don't know when. We do know that all possible start times $t \in [t_1, t_2]$ are equally probable.

The measurement sequence is pre-programmed and cannot be changed. It consists of n successive measurements which have timings $m_1, m_2, \ldots, m_n$ that are fixed. If t is the time that the probe starts the measurement sequence, the first measurement occurs at time $t + m_1$, the second at time $t + m_2$, and so on. The last measurement occurs at time $t + m_n$.

The measurements are instantaneous events — each happens in a very short time, so we consider the duration of any measurement to be 0 seconds.

The probe is rotating in space and cannot be controlled. Due to its rotation, there are intervals of time when measurement devices are pointed at the Sun. If the probe were to use a measurement device while the device is pointed at the Sun, the device's sensors would be destroyed and the whole probe would be lost. We do know the trajectory and the rotation of the probe and therefore there is a set of k time intervals $[b_1, e_1], [b_2, e_2], \ldots, [b_k, e_k]$ during any of which no measurement may be made.

Find the probability that the probe makes all measurements successfully and is not lost due to solar radiation damage!

Note that in this problem, all known times and time interval lengths are expressed as integers. However, the time of the start of the measurement sequence is unknown and we suppose that it may be expressed as any real number in the interval $[t_1, t_2]$.

Input

The first input line contains four integers n, k, t_1, and t_2. The value n represents the number of measurements, k represents the number of time intervals in which no measurement may be made, and t_1 and t_2 represent the time limits in which the measurement sequence can begin. The second input line contains n integers representing the sequence $m_1, m_2, \ldots, m_n$ of pre-programmed time moments in which measurements are made after the measurement sequence has begun. The sequence m_i is strictly increasing. Finally, there are k input lines which each have two integers b_j and e_j describing one time interval $[b_j, e_j]$ in which no measurement may be made. It's guaranteed that $b_j < e_j$, and the intervals do not overlap, i.e., $e_{j-1} < b_j$ for all $j > 1$.

We know that $1 \leq n \leq 10\,000$, $1 \leq k \leq 10\,000$, $n \cdot k \leq 10^7$, and $0 \leq t_1 < t_2 \leq 10^{16}$. All values $m_1, \ldots, m_n$ and $b_1, e_1, \ldots, b_k, e_k$ are non-negative and less than 10^{16}. All values are separated by single spaces.

Output

Output the probability of the probe's survival, that is, the probability that no measurement will be made during any of the intervals in which the Sun could damage the sensors. Your answer must be within an absolute or relative error of 10^{-6} of the correct answer.

Sample Input 1	Sample Output 1
2 2 10 20 1 5 12 14 15 18	0.4000000000

Sample Input 2	Sample Output 2
6 3 100 200 0 10 20 30 40 50 140 150 170 171 210 300	0.0900000000

Problem L
Suspension Bridges

Photo by Bertbau, cc-sa 4.0 int

Mountain villages like to attract tourists by building suspension bridges, such as the one depicted here in the Harz Mountains in Germany. These bridges allow adventurously-inclined people to seek their thrills by crossing over deep gorges. To make sure that everyone gets just the right amount of excitement, the sag at the deepest point of the bridge should be significant relative to the distance the bridge covers.

Given the distance between the anchor points where the bridge is attached, and given a desired amount of sag, compute how long each of the cables holding the suspension bridge needs to be!

To help you solve this task, here is some background: A free-hanging suspension bridge will take on the form of a catenary curve (*catena* is Latin for chain), just like a free-hanging chain between two poles. Given the horizontal distance d between two anchor points and the desired amount s the cable is sagging in the center, there exists a positive parameter a such that $a + s = a \cdot \cosh\left(\frac{d}{2a}\right)$. The length of the cable is then given by $\ell(a, d) = 2a \cdot \sinh\left(\frac{d}{2a}\right)$.

The functions $\sinh$ and $\cosh$ denote the hyperbolic sine and hyperbolic cosine, respectively, which are defined as follows:

$$\sinh x = \frac{e^x - e^{-x}}{2} \qquad \cosh x = \frac{e^x + e^{-x}}{2}$$

Input

The input consists of a single test case with two space-separated integers d and s given on a single line such that $0 < d \leq 1\,000$ and $0 < s \leq 1\,000$. The number d denotes the distance between the anchor points and s is the desired sag at the center of the bridge.

Output

Output the length of cable needed to cover the distance between the anchor points to achieve the desired sag. Your answer should be correct within an absolute error of 10^{-4}.

Sample Input 1	Sample Output 1
400 40	410.474747252

This page is intentionally left blank.

Problem M
Umbral Decoding

You are planning a clever attack against a new encryption algorithm. To succeed, you need to find the key, which is a pair of integers (p, q). We can think of the key as a point on a two-dimensional integer lattice whose location is unknown. However, you know that (p, q) lies in the square spanned by the lattice points $(0, 0)$ and (n, n) for a given n, i.e. $0 \leq p, q \leq n$.

Your attack has three stages:

1. Identify safe points and their bounds.
2. Eliminate as key candidates those points that lie in the "umbra" of any safe point.
3. Test the remaining points to see which one is the key.

Stage 1 has already been performed and you are given several safe points of the form (x, y, b) as input.

In Stage 2 you eliminate a point (p, q) if it lies in the umbra of any safe point. Point (p, q) is in the umbra of safe point (x, y, b) if and only if

$$|x - p|^3 + |y - q|^3 \leq b.$$

Your task in this problem is to count how many points will be left for testing in Stage 3 so that we have an estimate of the amount of work left to complete the attack.

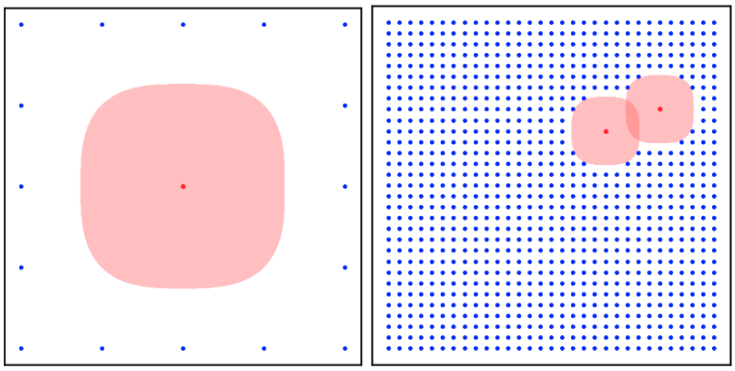

Figure M.1: Sample input safe points and umbra (red) and remaining points (blue).

Input

Input begins with two integers n, k on a single line, separated by a space, with $2 \leq n \leq 100\,000\,000$, and $0 \leq k \leq 100$. Following that are k lines each containing three integers x, y, b separated by spaces, representing the safe points. The values of x and y are both in the range $[0, n]$. The bound b lies in the range $[0, n]$.

Output

Output the number of points (p, q), with $0 \leq p, q \leq n$, that do not lie in the umbra of any safe point.

Sample Input 1	Sample Output 1
4 1 2 2 2	16

Sample Input 2	Sample Output 2
30 2 20 20 30 25 22 30	891