Problem A. Intentional Blank Page

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

Some contests copy the problem set double-sided (i.e., front and back). They do not, however, start a problem on a back page. If a problem requires an odd number of pages (1, 3, 5, 7, ...), they leave the last page for that problem (a back page) blank so that the next problem starts on a front page (they usually put the message "*This page is intentionally left blank.*" on that last page for the problem).

Given number of problems and number of pages for each problem, determine how many pages are left blank in the problem set, i.e., how many pages have the special message.

Input

The first input line contains an integer, p ($1 \leq p \leq 20$), indicating the number of problems in the set. Each of the next p input lines contains an integer (between 1 and 100, inclusive) indicating the number of pages for a problem.

Output

Print the number of blank pages.

Example

Input	Output
5 13 10 70 1 1	3
3 6 12 8	0
1 1	1

Problem B. Full House

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

A poker hand consists of 5 cards. The hand is called a *full house* when three cards match in value and the remaining two cards also match in value (but the second value is different from the first value).

Given an integer consisting of exactly five digits, determine if it is a *full house* (three digits one value, two digits another value).

Input

There is only one input line; it provides an integer between 10000 and 99999, inclusive.

Output

Print `YES` or `NO`, indicating whether or not the input integer is a *full house*.

Example

Input	Output
12121	YES
12345	NO
88888	NO
77555	YES

Problem C. Duck Stress Balls

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

The UCF Programming Team organizes an onsite contest for the high schools in Florida. This contest is modeled after ICPC and each team consists of three students. In the most recent contest, the contest organizers purchased some stress balls (ducks) to give away as swags. The ducks happen to be in several different colors and the organizers decided to give each team three ducks of different colors, i.e., the three students on a team will have ducks of different colors. The process became a little cumbersome so here you are to help!

Given number of duck colors and the count for each color, determine if the three students on each team can get ducks of different colors. Assume that there are $3n$ total ducks to be divided among n teams.

Input

The first input line contains an integer, c ($3 \leq c \leq 10$), indicating the number of different duck colors. Each of the next c input lines contains an integer (between 1 and 60, inclusive) indicating the number of ducks of a given color. Again, there will be a total of $3n$ ducks to divide among n teams.

Output

Print `YES` or `NO`, indicating whether or not each team can get three ducks of different colors.

Example

Input	Output
5 2 3 3 2 2	YES
3 59 1 60	NO
4 3 3 3 3	YES

Explanation

In the second Example, there are 120 ducks to be divided among 40 teams. It is not possible for each team to get three ducks of different colors.

Problem D. Twin Numbers

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

An integer is a *twin* if every digit at position $2n-1$ in the number is the same as the digit at position $2n$, e.g., 88, 8855, 775588, 8888. Note that a twin number must have an even number of digits.

Given a range (lower bound and upper bound), determine how many *twins* there are in that range.

Input

There are two input lines; the first line provides the lower bound for the range and the second line provides the upper bound. Assume that $10 \leq$ lower bound $\leq$ upper bound $\leq 10^{100000}$.

Output

Print the number of twins in the given range. Since the answer can be large, print the value mod 10^4+7.

Example

Input	Output
20 30	1
22 99	8
1000 4489	39
330210 1234567890	9683

Explanation

The number of twins for the last Example range is 19690 so the output will be 19690 mod 10007 = 9683.

Problem E. Simple House

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

You have purchased a piece of land that is rectangular. The land boundaries are not necessarily parallel to x and y axes. You want to build a rectangular house on your land but you'd like the house boundaries to be parallel to x and y axes. The house can touch the land boundaries but cannot go beyond the land. You would also like the house to look nice so the house must have a fixed ratio of its width (x-axis) to its height (y-axis).

Given the coordinates of the four corners of the piece of land, as well as two integers w and h (which represent the ratio of the width of the house to the height of the house), determine the area of the largest house that you can build.

Input

There are five input lines:

- Each of the first four input lines contains two integers, x and y $\left(-10^4 \leq x,\ y \leq 10^4\right)$, providing one corner of the land. The four corners are provided in a clockwise order, with the bottom-most, left-most corner first.
- The fifth input line contains two integers, w and h $(1 \leq w,\ h \leq 20)$, representing the ratio of the width to height.

Output

Print the area of the largest house that you can build. Any answer within an absolute or relative error of 10^{-6} of the correct answer will be accepted.

Example

Input	Output
3 0 0 4 8 10 11 6 2 1	12.5
0 0 0 5 100 5 100 0 20 1	500.0

Problem F. Pathfinder and Wolves

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

The famous explorer, Moose The Pathfinder (MTP), is at cell position $(1,\ 1)$ in an $r \times c$ rectangular maze and needs to get to cell position $(r,\ c)$. MTP can go in any of the four directions (north, south, east, west) except when in a border cell (the border cells have fewer move options since MTP cannot move across the outer walls of the maze). Unfortunately, wolves are hiding in some cells and, obviously, MTP wants to stay as far away as possible from these cells.

Given the maze, determine the closest MTP will have to get to any cell with wolves while trying to go from position $(1,\ 1)$ to position $(r,\ c)$. Note that MTP would travel longer if it helps to be farther away from the cells with wolves, i.e., MTP would take a longer path if it means being farther away from the cells with wolves. MTP distance to a cell with wolf is defined as the number of steps (in the four directions) from where MTP is to the cell with wolf.

Assume that there will be a path for MTP that will not go thru any cell with wolf, i.e., the closest MTP will have to get to a cell with wolf will be of distance 1. Also assume that there is at least one cell with wolf in each maze.

Input

The first input line contains two integers: r $\left(2 \leq r \leq 10^3\right)$, indicating the number of rows in the maze and c $\left(2 \leq c \leq 10^3\right)$, indicating the number of columns in the maze. The maze is provided in the following r input lines. Each of these input lines contains c characters starting in column 1. The cells with a wolf are indicated with the letter W and the empty cells are indicated with hyphen (-). Assume that the starting and ending positions will be empty.

Output

Print the closest MTP will have to get to any cell with wolf.

Example

Input	Output
5 5 ----W --W-W ----W ----- W----	2
3 5 ---WW ----- -----	2
3 3 --- --W ---	1
3 3 -W- --- ---	1

Explanation

In the third Example, MTP is of distance 1 to a cell with wolf when at destination.

In the last Example, MTP is of distance 1 to a cell with wolf when at the starting position.

Problem G. Festival of Blossoms

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

In the Kingdom of Floralia, the annual Festival of Blossoms marks the arrival of spring with a cherished tradition: every household must display a string of exactly N flowers on their door. Each year, the king announces K different flower colors available at the royal gardens, and citizens may use any combination of these colors to craft their decoration (a string of flowers).

This year's festival carries special significance. The royal oracle has emerged from the Temple of Petals with a prophetic vision: a sacred substring S of colors that promises to bring prosperity and good fortune to the kingdom. According to the prophecy, every flower-string must contain this blessed pattern (contiguous substring) somewhere along its arrangement, meaning the colors of S must appear consecutively and in the exact same order somewhere within the N flowers. Only then will the divine blessing reach each home in Floralia.

The king's assistant, a meticulous man who takes great pride in the festival's organization, faces his annual challenge. He firmly believes that every household deserves a unique decoration (a string of flowers), and no two families should receive identical arrangements on this special day. In his careful records, two strings of flowers are considered the same if each of the N flowers are identical in color when viewed left to right.

With this year's values of N and K announced, and the oracle's prophetic substring S revealed at dawn, the assistant turns to you with an urgent question: exactly how many distinct strings of N flowers containing S are possible?

The answer will determine whether the kingdom has enough unique designs for all its households to participate in this blessed tradition. Time is short and the festival begins at sunset.

Given N, K, and S, determine the number of unique flower strings which contain the sacred contiguous substring.

Input

There is only one input line; it contains two integers N $\left(1 \leq N \leq 10^4\right)$ and K $(1 \leq K \leq 26)$, followed by substring S $(1 \leq |S| \leq \min(N,\ 1000))$.

Assume that the flower colors are represented by the first K lowercase letters, e.g., if $K = 4$, the flower colors will be a, b, c, and d. The substring S is also made up of the first K lowercase letters.

Output

Print one integer, the number of unique strings of N flowers containing the sacred substring S. Since the answer can be quite large, your program should report the number as the smallest nonnegative remainder when divided by $10^9 + 7$.

Example

Input	Output
4 2 a	15
2 3 ca	1
10 4 abc	127248
5 26 wow	2027
10 26 wow	176416529
31 2 a	147483633

Problem H. Birthday Bash

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

Bob loves birthdays. In fact, he loves them so much that he's taken his celebrations interplanetary. Whenever he lands on a new planet, his first mission (before food, water, or breathable air) is to throw the biggest birthday bash the galaxy has ever seen. The catch? Bob only parties when every single birthday is represented.

On this planet, the year consists of exactly n days, and each person Bob meets has an equal and independent chance of being born on any of those days. Starting from an empty guest list, Bob adds people to his party one by one, selecting each guest uniformly at random from the population. He stops as soon as his guest list includes someone born on *every* of the n possible birthdays.

What is the **expected number of people** Bob needs to invite until he has a complete set of birthdays?

Given an integer n, find the expected number of randomly selected people needed until every one of the n possible birthdays is represented at least once. Each person's birthday is uniformly and independently distributed across the n days. Calculate this answer mod $10^9 + 7$.

Input

There is only one input line; it provides an integer n $\left(1 \leq n \leq 2 \cdot 10^9\right)$, the number of days in a year on this planet.

Output

Print the expected number of people mod $10^9 + 7$. If the answer is a fraction, output it as an integer in the form $\left(A \cdot B^{-1}\right)$ mod $(10^9 + 7)$, where B^{-1} is the modular inverse of B mod $(10^9 + 7)$.

Example

Input	Output
1	1
2	3
6	900000021
10	289682571
14	980613883
20	26325398
1075	742969056
99998	924541619

Explanation

For the Example Input $n = 6$, we will have:
$A = 147$
$B = 10$
Answer $= \left(A \cdot B^{-1}\right)$ mod $(10^9 + 7) = 900000021$

For the Example Input $n = 20$, we will have:
$A = 279175675$
$B = 3879876$
Answer $= \left(A \cdot B^{-1}\right)$ mod $(10^9 + 7) = 26325398$

Problem I. Dark Things

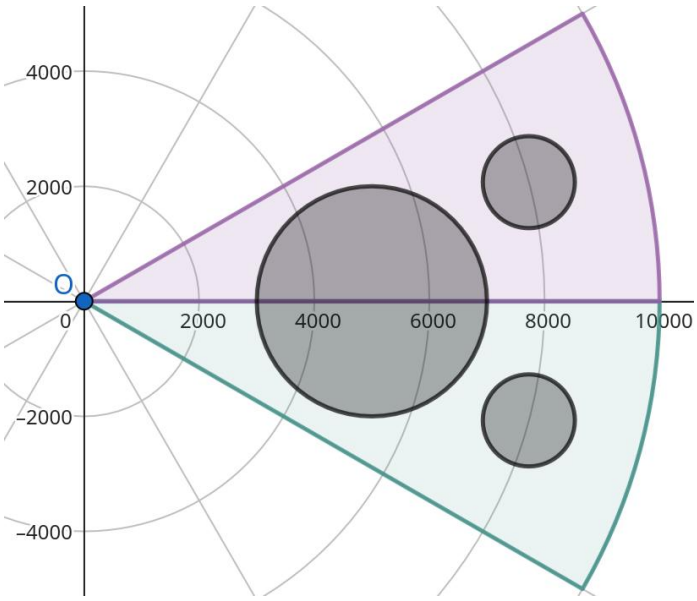
Source file name:	Dark.c, Dark.cpp, Dark.java, Dark.py
Input:	`Standard`
Output:	`Standard`

As you sit in the observatory gazing upon the stars, several alerts go off drawing your attention to the main console which reads:

```
Warning: Several unidentified objects spotted!
```

These objects are quite peculiar, not matching any physical properties of any known or theorized substance or energy, not even dark matter or dark energy! Of course, your first thought is about how many great papers this could lead you to publishing and just like many great physicists, you started by deciding to give these objects a perfectly apt name: "Dark Things!".

Now that you have named these objects, you have decided to focus on the less important details, such as how the dark things interact with surrounding masses and energy. To study these in further detail, you decide to employ k sensors on the observatory's telescope systems and you want each of these sensors to do an equivalent amount of work. To determine how to allocate the search space of each sensor, you have decided to model your situation using geometry as illustrated below:



In the cartesian plane, you model the Earth as a point in the plane located at the origin. At the current point in time, you know that any point which is within R parsecs from earth and is at most θ degrees away from the positive x-axis can be seen by the observatory. The dark things can also be modelled as non-intersecting circles whose centers are r_i parsecs away from the earth, are φ_i degrees away from the positive x-axis, and have a diameter of d_i. Due to the limitation in the observatory's technology, you also know that each dark thing must be fully contained within the set of observable points.

Each sensor will scan all points that are not contained within any dark thing, are within R parsecs of the earth, and are within some range of angles measured from the positive x-axis. To make sure all sensors do an equal amount of work, you must find $k-1$ angles ψ_j such that the first sensor scans from $-\theta$ to ψ_1, the second sensor scans from ψ_1 to ψ_2, the third sensor scans from ψ_2 to ψ_3, and so on until the last scanner which scans from ψ_{k-1} to θ and the areas of the set of points each sensor scans are equal.

Given the region of observable points and the location of each dark thing, partition the search space into k equal areas, where any point within a dark thing is ignored.

Input

The first input line contains four integers: n $(1 \leq n \leq 10^3)$, indicating the number of dark things, k $(2 \leq k \leq 50)$, indicating the number of sensors, R $(10^4 \leq R \leq 10^6)$, indicating the maximum distance of any observable point, and θ $(30 \leq \theta \leq 60)$, indicating the maximum angle of any observable point when measured from positive x-axis.

Each of the following n input lines will contain three integers: r_i $(1 \leq r_i \leq R)$, φ_i $(|\varphi_i| \leq \theta)$, and d_i $(1 \leq d_i \leq 2000)$, indicating the location and size of the i^{th} dark thing. It is guaranteed that no two dark things touch or intersect and that each dark thing is fully contained within the set of observable points.

Output

Print $k-1$ floating point numbers, one per output line, indicating the angle of each partition. Any answer within an absolute or relative error of 10^{-6} will be accepted.

Example

Input	Output
3 2 10000 30 5000 0 2000 8000 15 800 8000 -15 800	0.0000000000
3 3 10000 60 9000 55 50 9500 -59 10 9250 57 30	-20.0009600000 19.9979900000

Explanation

The image on the previous page corresponds to the first example, where we are splitting the observable set of points into two equal regions (the green and purple sections). In this scenario, both sensors will cover an area of roughly 5.0286×10^7 parsecs^2.

Problem J. Galactic Voyage

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

You are the navigator aboard the starship Celestial Wanderer, bound for Sagittarius A^*, the supermassive black hole at the galaxy's center. Your vessel is equipped with a state-of-the-art Quantum Warp Drive. Unlike conventional propulsion systems that provide continuous thrust, this warp drive operates through discrete spatial translations. The drive has been calibrated with S distinct warp configurations, where each configuration can transport your ship exactly s_i astronomical units forward along your trajectory. This drive only works while traversing towards the center of the galaxy, so you'll have to equip a different drive for the return trip.

In the observation lounge, you overhear a group of veteran explorers discussing an ancient network of wormholes that permeates the galaxy along radial paths toward its center. According to these explorers, there are P distinct series of wormholes along the straight-line path from our current location to the center of the galaxy. Each of the P series forms a regular pattern of wormhole entrances, with wormholes in series i being located at distances of 0, p_i, $2p_i$, $3p_i$, ... astronomical units from your current position. It just so happens we are at a location where all P series converge.

The group of veteran explorers went on to detail exactly how these wormholes work. You can enter any wormhole from any series and exit from any wormhole in any series (including the same one), with two critical restrictions. First, you can only travel forward toward the galactic center, meaning if you enter a wormhole at distance X then you must exit at a distance greater than X. Second, gravitational distortions limit each wormhole jump to at most K astronomical units, meaning the distance between your entry and exit points cannot exceed K.

Mission control plans to place a resupply checkpoint station along this heavily traveled route to Sagittarius A^*. To ensure the station is accessible to all vessels using the same warp drive as yours, they want you to calculate the maximum distance D that you cannot reach from your starting point.

Given the ship's warp configurations S, the set of wormhole series P, and the value K in astronomical units, determine the maximum distance D from your current location that you cannot reach via some combination of warp drive and wormhole traversals.

Input

There are three input lines:

- The first input line contains three integers: S, P, and K ($1 \leq S, P, K \leq 10$), indicating (respectively) the number of warp configurations, the number of wormhole intervals, and the value of K.
- The second input line contains S integers, s_1, s_2, ..., s_s $\left(1 \leq s_i \leq 10^4\right)$, the set of warp drive distances.
- The third input line contains P integers, $p1$, $p2$, ..., p_p $\left(1 \leq p_i \leq 10^4\right)$, the set of wormhole intervals.

Output

Print the distance D, in astronomical units, representing the maximum unreachable distance. If all locations are reachable or there are infinitely many unreachable locations, print `-1` instead.

Example

Input	Output
2 2 4 3 7 4 11	5
3 2 1 6 9 15 4 10	-1
3 2 5 8 17 20 10 15	63

Explanation

Explanation of the first Example Input/Output:

$S = \{3, 7\}$: This means that from your starting position 0, you can advance to 3, 6, 9, ...; you can also advance from your starting position 0 to 7, 14, 21, ... Additionally, you can combine these forward steps in any order, e.g., from starting position 0 you can traverse 3 steps followed by 7 steps to reach position 10.

$P = \{4, 11\}$: This means that there are wormholes at position 0, 4, 8, ... as well as at positions 0, 11, 22, ... With $K = 4$, you can traverse from the 0 portal to position 4, then from 4 to 8, then from 8 to 11; you can go from 8 to 11 because the distance between your entry point (8) and your exit point (11) does not exceed K (4). Note that you cannot traverse from the 0 portal to position 11 directly (even though P contains 11) because the distance between your entry and exit points exceeds K.

The largest value which is completely unreachable via some combination of warp configurations in S or wormholes in P is 5 (this means we can reach 6, 7, 8, 9, 10, ...).

Notes:

- When at a position (regardless of how you got there), you can use steps in S to advance. For example, after entering a wormhole, you can use steps in S to advance next.
- The values in P cannot be used as steps to advance, i.e., wormhole values cannot be used as steps. For example, in the first Example above, let's assume you move to position 7; from there you cannot use 11 (a value in P) to advance to position 18 (of course, there may be other ways to get to position 18).
- You can advance from one wormhole position to any other wormhole position (assuming the constraint K is met).

Problem K. Degenerating Walkways

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

You are exploring an old abandoned space station looking for a way to get home after your spaceship malfunctioned. The space station consists of n hubs and $n-1$ bidirectional walkways. Each bidirectional walkway connects two different hubs and any pair of hubs are directly or indirectly connected to each other.

After quite a bit of time exploring the station, the space station's power generators activate, revealing a Sustained Technological Astro-Relic (STAR) in each hub. There is exactly one STAR (an integer) in each hub. These STARs are just what you have been looking for, as you can use them to stabilize some portal technology you brought from your spaceship and will help you to get home! More specifically, you will attach (append) these STARs to your portal following the pattern (order) in a target sequence. The task is simplified by having no duplicate values in the target sequence, i.e., you know that $x_i \neq x_j$ in the target sequence for any indices i and j such that $i \neq j$.

STARs come in one of k forms (k different integer values) and from your engineering background, you know that certain sets of STARs can be attached to portal technology resulting in various stabilities. More specifically, if the i^{th} STAR appended to the portal is of form x_i (i.e., integer value x_i) and this value matches the next item in the target sequence, the stability of the portal increases by one. Note that the target sequence dictates what can be attached to the portal. If the i^{th} item in the target sequence is x_i, the i^{th} item attached to the portal must be x_i; taking any other STAR will result in the complete decay of the portal.

Your goal is to now collect STARs to stabilize your portal technology and travel home. Due to being really bad at directions, you are not sure where (which hub) you are in the space station, however you do know what the layout of the space station is, so to collect a STAR set, you are going to randomly traverse the station.

Unfortunately, the power being reactivated has made traversing the space station more difficult. Due to the age and the lack of maintenance of the walkways, traversing any walkway will result in it degenerating and it will no longer be able to be used again, i.e., each walkway can be used at most once. Thus, you will collect STARs with the following strategy:

1. In the current hub, take the STAR in the hub if it is of the form you need, i.e., matches the next item in the target sequence.
2. If you have collected t total STARs so far, you attempt to use the portal. The portal will work with a probability of success of $1 - \left(\frac{p}{q}\right)^{t+1}$, where p and q are given constants.
3. If Step 2 does not succeed, traverse to a new hub. Of the walkways that are currently functioning, choose one equiprobably. If there are no walkways connected to the hub you are currently in, you will stay in the current hub and repeat Step 2 (i.e., attempt to use the portal again) until you succeed, i.e., the portal successfully brings you home.

You now want to determine what the expected stability of the portal will be when you take it home. Since you are unsure of which hub you are currently in, you will assume that each hub has an equal probability of being the starting hub.

Given the space station layout, the target sequence of STARs to be collected, and which type of STAR each hub contains, determine the expected stability of a portal that can be created.

Input

The first input line contains five integers: n $(1 \leq n \leq 10^5)$, indicating the number of hubs in the space station, m $(1 \leq m \leq 10^5)$, indicating the size of the target sequence of STARs, k $(1 \leq k \leq 10^9)$, indicating the number of types of STARs, and p and q $(0 < p < q < 998,244,353)$, indicating the given constants used for computing the probability of success.

The second input line provides the target sequence, i.e., this input line will consist of m integers where the i^{th} integer corresponds to the i^{th} target STAR. Each of these integers will be between 1 and k, inclusive, and no pair of integers will be equal.

The next input line will be composed of n integers, where the i^{th} integer indicates the STAR type in the i^{th} hub. Each STAR type will be denoted by an integer between 1 and k, inclusive. Note that different hubs may have the same STAR type.

Each of the following $n-1$ input lines will contain two integers u and v $(1 \leq u, v \leq n)$ denoting that there is a walkway connecting hub u and hub v. Note that the space station is completely connected, i.e., you can reach any hub from any other hub.

Output

Print the expected stability of the portal after successfully using it. Since the answer can be represented as $\frac{a}{b}$ where a and b are integers, output $a(b^{-1}) \mod 998,244,353$, where b^{-1} denotes the multiplicative inverse of $b \mod 998,244,353$.

Example

Input	Output
2 2 2 1 2 2 1 1 2 1 2	124780545
4 5 6 4 5 3 5 2 1 6 5 2 3 2 1 2 1 3 1 4	343422678

Explanation

The answer to the first example corresponds to the fraction 7/8 and the answer to the second example corresponds to the fraction 26947/37500.

Problem L. Copogonia

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

Copogonia is a country with a set of n cities, which currently has n roads. The set of roads forms a convex polygon, hence the name of the country. Unfortunately, some of the citizens have been complaining about traveling times between cities. Due to the limited road system, there are always precisely two ways to travel between any pair of distinct cities (clockwise or counter-clockwise), and for some pairs of cities, even the shorter of the two possible routes seems like it takes too long.

After many rounds of careful surveys, it's been determined that the citizens would be happy if no individual travel distance between a pair of cities exceeded length m, i.e., the path between any two cities has length m or less.

The government of Copogonia has done extensive research and decided that they could potentially build up to k new roads where each road directly connects two cities that were not previously connected via a direct road with a distance equal to the straight line distance between the two cities. Each road, however, comes with a cost of development. Naturally, the government of Copogonia would like to minimize the cost of development while keeping all of its citizens happy. They have tasked you with finding the precise subset of roads to develop to achieve their goals.

Given the locations on the Cartesian plane of the n cities of Copogonia (in the order they are connected with roads, noting that there is also a road from the last city to the first), a list of k potential additional roads that could be added (pair of cities and cost to build the road), and the maximum distance, m, that citizens are willing to travel on a single trip between pairs of cities, determine the minimal cost of developing a subset of roads that will keep the citizens of Copogonia happy.

Input

The first input line contains three space separated integers: n ($4 \leq n \leq 50$), indicating the number of cities in Copogonia, k ($1 \leq k \leq 10$), indicating the number of potential additional roads to build, and m, indicating the maximal travel distance between any pair of cities that the Copogonians are willing to tolerate.

Each of the next n input lines contains two space separated integers, x_i and y_i ($0 \leq x_i,\ y_i \leq 10^4$, $1 \leq i \leq n$), indicating the location of city i on the Cartesian grid (assume that no two cities are at the same location). There is a straight line road connecting city i to city $i+1$ ($1 \leq i \leq n-1$) and a road connecting cities n and 1. Roads can be travelled in either direction.

The potential roads that can be added are provided in the next k input lines. Each of these k input lines contains three space separated integers: u, v ($1 \leq u, v \leq n,\ u \neq v$) and c ($1 \leq c \leq 10^8$), indicating that a road could be added between city u and city v at a cost of c. It's guaranteed that there's currently no road from city u to v and that no two of the potential roads listed will be between the same pair of cities.

Assume that:

- The input is such that at least one road must be added, i.e., with no extra roads the citizens are not happy (at least one of the shortest paths is greater than length m).
- If all of the potential roads are built, the citizens will be happy, i.e., all of the shortest paths will have length m or less.
- The value of m will be such that adding or subtracting 0.001 to m will not modify the answer.

Output

Print the minimum cost of a set of roads to build that would keep all the citizens of Copogonia happy.

Example

Input	Output
4 2 15 0 0 0 10 10 10 10 0 1 3 1000 2 4 500	1500
5 3 2003 0 0 0 100 1000 100 2000 99 1000 0 2 4 50 4 1 123 3 5 47	170

Problem M. Weekend Gardening

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

Your roommate is planning some weekend projects to make your yard look better. The plan is to buy some plants. Your roommate wants to spend at least L dollars and at most H dollars. There are three different types of plants: cheap, normal, and expensive. Unfortunately, you cannot tell the difference by just looking at a plant. You know that each type has a different cost, and you know how many of each type your local nursery has in stock.

Now you are spending your time walking through the nursery aisles, picking out a plant, bringing it over to be rung up by the clerk, and praying that you stay within your desired total cost (your allotted budget). If you are not within your desired total cost (i.e., if your total cost is too low or too high), your roommate will get frustrated.

Given the cost and number of each plant type along with your desired budget range, determine the probability that you can buy plants one at a time and staying within your total cost range.

Input

The first input line contains two integers: L and H ($1 \leq L \leq H \leq 10^9$), representing the low and high end of the budget, respectively.

The second input line contains three integers: C, N, and E ($1 \leq C < N < E \leq 10^9$), representing the cost of the cheap, normal, and expensive plants, respectively.

The third input line contains three integers: c, n, and e ($1 \leq c, n, e \leq 100$), representing the number of cheap, normal, and expensive plants, respectively.

Output

Print the probability that you will successfully complete the purchase within the desired budget by choosing plants one at a time. Your answer will be accepted if it is within 10^{-6} of the correct answer.

Example

Input	Output
30 60 5 15 70 3 1 1	0.2
40 60 5 15 70 3 1 1	0.0