Problem A. A Non-Palindromic Modification

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

A sequence a of length n is called *palindromic* if $a_i = a_{n-i+1}$ for all $1 \le i \le n$.

You are given the sequence b of n integers $b_1, b_2, \ldots, b_n$. Determine whether it is possible to increase **exactly one** element of b by 1 in such a way that the resulting sequence will **not** be palindromic.

Input

The first line of the input contains one integer n ($1 \le n \le 1000$) — the length of the input sequence. The i-th of the following n lines contains one integer b_i ($1 \le b_i \le 1000$) — the i-th element of sequence b.

Output

If it is possible to increase exactly one b_i by 1 and get the sequence that is not palindromic, print `1` in the only line of the output. Otherwise, print `0`.

Examples

standard input	standard output
1 1	0
2 20 21	1

Problem B. Gleb and Liteyny Avenue

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Gleb has just had a good time in a bar on Nekrasova street, and is now heading to the place of his accommodation that happened to be hotel Indigo. His path goes along Liteyny Avenue that we consider to be a perfect straight line.

For the purpose of this problem Gleb only moves along the avenue and across it. He now stays at point 0 along the avenue and the hotel he goes to is located at point L. Moreover, the hotel is located on the other side of the avenue. There are n pedestrian crossings between Gleb and Indigo, the i-th of them is located at integer point x_i. In order to get to the hotel Gleb needs to walk all the avenue from point 0 to point L and cross the street at some moment of this walk.

All crossings are equipped with a traffic light. All these traffic lights have cycles of the same duration. They go green for g seconds and red for r seconds. However, they are not synchronized, i.e. each traffic light has some random equiprobable cycle shift. All shifts are integers, so there are only $r + g$ possible shifts in total. Gleb knows integers r and g but at beginning he knows nothing about the shifts.

Each crossing traffic light has a timer that shows the current state of this traffic light's cycle. If the light is green, the timer shows how many seconds are left to cross the street, if the light is red, the timer shows how many seconds to wait before the light goes green. Gleb is able to notice the light's color and the timer state only when he is at the exact point this crossing and traffic light are located. The good part is that Gleb is capable of remembering everything he has seen during this walk, so he keeps in mind the current state of all the traffic lights he has already seen.

All coordinates are integer and are given in meters. Gleb moves with a speed of one meter per second. Moreover, it is known that pedestrian crossings are not too frequent. Formally, there are no three crossings located within $g + r$ meters.

Gleb can move in any direction along the avenue and cross it arbitrary number of times. It takes b seconds to cross the avenue, and the traffic light must be green during all this period.

Input

The first line of the input contains integers n, L, g, r and b ($1 \le n \le 100\,000$, $1 \le L, g, r \le 10^9$, $1 \le b \le g$) — the number of crossings along the avenue, initial distance between Gleb and hotel Indigo in meters, the duration of the green phase of the traffic light in seconds, the duration of the red phase of the traffic light in seconds, the time it takes Gleb to cross the avenue.

Then follow n lines that describe crossings' positions $x_1, x_2, \ldots, x_n$ ($0 < x_1 < x_2 < \ldots < x_n < L$). It is guaranteed that $x_{i+2} - x_i > g + r$ for all $1 \le i \le n - 2$.

Output

Print one real value — the expected time it will take Gleb to get to Indigo in case he acts optimally. The absolute or relative error should not exceed 10^{-9}.

Examples

standard input	standard output
1 10 2 4 1 2	12.6666666666667
2 820 30 50 23 400 810	866.0250000000000
3 100 20 50 1 10 15 85	107.6141690962099

Problem C. Socks Drying

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	512 megabytes

Gleb has socks of n distinct colors. In particular, he owns a_i pairs of socks of color i. He wears these pairs one after the other till there are no more clean pairs of socks left, then it is time for The Great Wash.

Gleb puts all of his socks into a washing machine with a drying option and waits for an two-three hours. Now, the worst part begins. Gleb needs to assemble socks in pairs again. To simplify the process we assume that all socks of the same color are indistinguishable. In other words, any two socks of the same color can form a pair. Obviously, Gleb never forms a pair of socks from two of distinct color.

Gleb performs the following process. We denote the set of all socks inside the washing machine as A and the set of all socks that haven't been matched yet and lie near Gleb as B.

1. Gleb puts his hand into a washing machine and checks whether there are any socks left in A. If there are none, the process finishes. Otherwise, he puts out one random sock (with equal probabilities for all socks in A), we denote this sock as x. This is a single action that takes exactly one second.
2. He then tries to find a pair for this sock among all socks with no pair already extracted out of the washing machine and lie near him, i.e. all socks in set B.
3. Gleb considers all socks of B in random order. All orders are equiprobable. It takes exactly one second to consider one sock from B. If the sock x matches the sock just extracted from A by color, they form a pair and this step finishes.
4. If all socks from B were considered and no match for x was found, Gleb puts x in the heap of unpaired socks. In other words sock x goes to set B.
5. Gleb goes back to step one.

While waiting for wash-and-dry cycle to finish Gleb wonders, what is the expected time the process described above will take? The last time he took part in a programming competition was several years ago, so he needs your help to find out the answer.

Input

The first line of the input contains a single integer n ($1 \le n \le 200\,000$) — the number of distinct colors of socks Gleb possesses.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 5$), the i-th of them denoting the number of pairs of socks of color i in Gleb's wardrobe.

Output

Print one real value — the expected number of seconds it will take Gleb to complete the whole process if he follows the steps given. Your answer will be accepted if its absolute or relative error does not exceed 10^{-6}.

Examples

standard input	standard output
1 1	3.000000000
1 3	9.000000000
2 1 1	7.000000000
3 3 2 2	29.571428571

Problem D. Christmas Children Circle

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

There are n children numbered $1, 2, \ldots, n$. They stand in a circle. Each child has a bag with non-empty set of distinct positive integers.

As a part of Christmas celebration, children want to play a game of the following rules:

1. Each child has to take exactly one integer from her bag.
2. Each child passes her integer to the next one, so child 1 passes her number to child 2, child 2 passes her number to child 3, and so one, child n passes her number to child 1.
3. Each child puts her newly acquired number back in her bag.
4. The procedure described above should be performed exactly once. Children win if after they are done with passing integers, each bag contains only distinct integers, i.e. no bag contains the same integer twice.

Help children win the game or find out that this is impossible.

Input

The first line of the input contains a single integer n ($2 \le n \le 500\,000$), the number of children.

The i-th of the following n lines contains the description of the i-th child' bag. The first integer of each description is k_i ($1 \le k_i$) — the number of integers in the bag. Then follow k_i integers a_{ij} ($1 \le a_{ij} \le 10^9$), which describe the content of the bag.

The total size of all bags is constrained as $\sum k_i \le 500\,000$.

It is guaranteed that all integers in each particular bag are distinct.

Output

If there is no way for children to win the game, print `-1` in the only line of the output. Otherwise, print n integers, $x_1, x_2, \ldots, x_n$, where x_i is the integer that the i-th child must pull out of her bag and pass to the next child in order to succeed in the game.

Example

standard input	standard output
3 2 1 4 2 4 3 2 3 1	1 4 1

Problem E. Construct The Integer

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

For any positive integer y we define $A(y)$ as the set of all positive integers that are anagrams of y. Formally, we consider the sequence of all digits of y in decimal representation without leading zeroes, then we apply all possible permutations to this sequence. For any permutation we throw away leading zeroes and assemble digits back to an integer. All integers that can be obtained that way belong to $A(y)$. For example, for 2021 set $A(2021)$ consists of integers 122, 212, 221, 1220, 2120, 2210, 1202, 2102, 2201, 1022, 2012 and 2021. Note, that set $A(y)$ always contains y, thus it is never empty.

For any positive integer x we define $S(x)$ as the set of all positive integers z such that greatest common divisor of all integers in $A(z)$ equals x.

You are given the integer n, find the **minimum** integer z in $S(x)$, or determine that $S(x)$ is empty.

Input

The first line of the input contains one integer t ($1 \le t \le 50$) — the number of the test cases.

Each test case data consists of one integer n ($1 \le n \le 10^{18}$).

Output

For each test case, print one integer on a separate line. If the corresponding set $S(n)$ is empty, print `-1`. Otherwise, print minimum element of $S(n)$.

Example

standard input	standard output
2 12 2021	48 -1

Problem F. Build the Non-Cactus

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

In graph theory, a *cactus* is a connected undirected graph in which any two simple cycles have at most one vertex in common.

Given an integer n, build a **connected** graph with n vertices that is **not** a cactus. Note that your graph can't have self-loops or multiple edges. The number of edges in your graph should be minimum possible.

Input

The input consists of a single integer n ($2 \le n \le 1000$).

Output

If there is no connected graph of n vertices without self-loops and multiple edges that is not a cactus, print `-1` in the only line of the output. Otherwise, first print positive integer m — the number of edges in your graph. Then print m lines, each containing two integers — edges of the resulting graph. Use consecutive integers $1, 2, \ldots, n$ to enumerate the vertices of the graph.

If there are more than solutions with minimum number of edges, print any of them.

Example

standard input	standard output
3	-1

Problem G. Double Elimination

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

One of the greatest e-sport events, The Invitational TOAD-3 Championship, will feature 16 best teams in the world. In order to prolong the fascinating event and acquire more money from advertising a double-elimination tournament scheme will be used.

First, all teams are ordered by their rating and the i-th team in the sorted list plays versus the team at position $(17 - i)$. The winner of each of these matches goes to the winners' bracket, the loser goes to the losers' bracket.

Participants of the winners' bracket continue to play a single-elimination tournament, except that the losers of each round "drop down" into the losers' bracket.

Each round of the loser's bracket is conducted in two stages; a minor stage followed by a major stage. Both contain the same number of matches which is the same again as the number of matches in the corresponding round of the winners' bracket. If the minor stage of a looser's bracket round contains 2^k matches, there will be 2^k winners. Meanwhile, the 2^k matches in the corresponding round of the winners' bracket will produce 2^k losers. These 2^{k+1} teams will then form pairs for 2^k matches of the corresponding major stage of the looser's bracket. At this stage of the round, looser of the winners' bracket round can play only against winners of the minor stage of the looser's bracket.

In both brackets, before each round opponents are chosen at random to form pairs. Same apply to the process of forming pairs of type "loser from winners' bracket versus winner of the minor stage from the losers' bracket" at the major stages of the losers' bracket rounds.

All players who are eliminated from the same stage are considered to share the same position in the resulting ranking.

Below is the scheme of how the double-elimination tournament for 16 teams is held.

- The first round is played; winners of the first round form the winners' bracket, losers of the first round form the losers' bracket.
- The eight losers of the first round form pairs of the first minor stage of the losers' bracket, we denote this stage as L_8a. The four losers are eliminated (their participation in the tournament is over, they share places from 13 to 16), while the four winners proceed to the major stage of this round, we denote it as L_8b.
- The eight winners of the first round form pairs for the first round of the winners' bracket, we denote this as W_8. The four winners proceed to the stage W_4, four losers proceed to stage L_8b.
- During stage L_8b, each losing team from W_8 plays against the winning team from L_8a, the four losers are eliminated (they end up at places $9 - 12$), while the winners proceed to the stage L_4a.
- The four winners of L_8b go to stage L_4a. The losers are eliminated (they end up at places $7 - 8$), while two winners proceed to stage L_4b.
- The four winners of W_8 form pairs for the second round of the winners' bracket (named W_4), the two winners proceed to stage W_2, the two losers proceed to stage L_4b.
- In stage L_4b, each losing team from W_4 plays against some winner L_4a, the two losers are eliminated (they end up at places $5 - 6$), while the winners proceed to stage L_2a.
- In stage L_2a the two winners of L_4b play one match against each other, the loser finishes the tournament at the 4-th place, the winner proceeds to stage L_2b (also know as loser's bracket final).

- In stage W_2 one match is played. The winner proceeds directly to the grand final, the loser proceeds to stage L_2b.
- In stage L_2b one match is played, the winner proceeds to the grand final, the loser finishes the tournament at the third place.
- The winner of the match in grand final is declared a champion, the loser finishes the tournament second.

You are given the list of teams with the results of all matches played by this team during the tournament in chronological order. Determine the final place for each team.

Input

The input contains 16 lines.

Each line consists of two strings — the team name t made up of lowercase English letters ($1 \le |t| \le 50$) and the string G with the outcomes of all games this team has played. String G consists of digits `1` and `0` only. If the i-th character of this string equals `1` it means that this team won its i-th match on the tournament. If the i-th character of this string equals to `0`, it means that this team lost its i-th match on the tournament.

You may assume that all team names are distinct and that the distribution of wins and losses correspond to some correctly held double-elimination tournament for those teams.

Output

Print 16 lines. Each line should contain the notation for place: one integer if the place is not shared, and two integers separated by single '-' — the highest and the lowest if the place is shared, followed by the team name.

Teams shall be listed by their places in increasing order. If several teams share a place, sort those teams by the name lexicographically. See sample output for further clarification.

Example

standard input	standard output
escmraeett 11100 neigulsievse 010 noet 1010 nduymianegt 00 psimraiett 10111111 ustprriov 1100 yccnrieuwq 010 go 1010 agmiicnigv 10110 ctosaasetb 00 sugtacmiivnngi 11010 hpaenlte 00 lggsdp 11110 taincf 010 ainlclea 010 asmtaeert 00	1 psimraiett 2 lggsdp 3 escmraeett 4 sugtacmiivnngi 5-6 agmiicnigv 5-6 ustprriov 7-8 go 7-8 noet 9-12 ainlclea 9-12 neigulsievse 9-12 taincf 9-12 yccnrieuwq 13-16 asmtaeert 13-16 ctosaasetb 13-16 hpaenlte 13-16 nduymianegt

Problem H. Roman Palindromes

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

Given the string S, consisting of letters '`I`', '`V`', '`X`', '`L`', '`C`', '`D`' and '`M`'. Your task is to split the string to the **minimum** number of strings that are:

- correctly written Roman numerals;
- palindromes.

The following table from Wikipedia displays how the Roman Numerals are written:

Individual decimal places	Thousands	Hundreds	Tens	Units
1	M	C	X	I
2	MM	CC	XX	II
3	MMM	CCC	XXX	III
4		CD	XL	IV
5		D	L	V
6		DC	LX	VI
7		DCC	LXX	VII
8		DCCC	LXXX	VIII
9		CM	XC	IX

Note that:

- The numerals for 4, 9, 40, 90, 400 and 900 are written using "subtractive notation" where the first symbol is subtracted from the larger one (for example, for 40 ("`XL`") '`X`' (10) is subtracted from '`L`' (50)). These are **the only** subtractive forms in standard use.
- A number containing several decimal digits is built by appending the Roman numeral equivalent for each, from highest to lowest.
- Any missing place (represented by a zero in the place-value equivalent) is omitted.
- The largest number that can be represented in the Roman notation is 3,999 (MMMCMXCIX).

Input

The first line of the input consists of one integer n — length of the string ($1 \le n \le 100\,000$).

The second line contains the string of length n, consisting of letters '`I`', '`V`', '`X`', '`L`', '`C`', '`D`' and '`M`'.

Output

In first line of the output print one integer k — the minimum number of correct roman palindromes that can form the given string being concatenated. Then print k lines, each line containing one string — a correct Roman numeral that is a palindrome, such as the concatenation of all k lines in the order of output is equal to the given string.

If there is more than one solution, print any of them.

Example

standard input	standard output
5 MMXXI	3 MM XX I

Problem I. 1%-Euclidean

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

Given n points $p_1, p_2, \ldots p_n$ in the two-dimensional Euclidean plane, calculate the total pairwise Euclidean distance between them. Simple as that.

Recall that Euclidean distance between two points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is defined as follows:

$$\text{dist}(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Input

The first line of input contains a single integer n ($1 \le n \le 500\,000$), the number of points.

The i-th of the following n lines contains a pair of integers x_i, y_i ($-10^6 \le x_i, y_i \le 10^6$), the coordinates of p_i.

Output

Output the value of $\sum_{i<j} \text{dist}(p_i, p_j)$. Your answer will be considered correct if its relative or absolute error does not exceed 10^{-2}.

Examples

standard input	standard output
3 -1 2 2 2 -1 -2	12.0000000000
4 0 0 2 0 0 2 2 2	13.6568542490

Note

In the first sample test the pairwise distances are 3, 4 and 5.

In the second sample test the pairwise distances are $2, 2, 2, 2, 2\sqrt{2}, 2\sqrt{2}$.

Problem J. Spectacular Ending

Input file:	`standard input`
Output file:	`standard output`
Time limit:	6 seconds
Memory limit:	512 megabytes

The game "Dungeons and Rainbows" uses dices of various shapes and various number of faces. The game process consists of moving between dungeons and performing various actions. The game has n states in total, that number already includes all possible combinations of positions, items, skills and so on. The game rules define for each state which particular dice should be used in order to determine the next state. As the dice being used can have more than n faces, rules of the game define how many faces of the dice should correspond to each particular outcome (transition to some other state).

For example, consider there are 3 states in a game and for one of them it is required to use a dice with 5 faces. One possible distribution can be the following: 2 faces correspond to transition to the state 1, 1 face corresponds to transition to state 2 and remaining 2 faces correspond to transition to state 3.

Players start the game in state 1 and the game lasts k rounds. In order to get a spectacular ending of the game players must make a transition to state s in the last round. Note, that it doesn't matter whether they have already visited this state in other rounds or not.

Experienced host knows probabilities of each face of every dice. In **each round** he chooses which faces will correspond to which outcomes. So, at the beginning of every round the host assigns the next state for face of the dice that will be used in this round (this is determined by current state only). If it happens that players transit to the same state more than once, the host can make a different assignment of outcomes to faces every time.

The host wants to maximize the probability that the game will have a spectacular ending. Calculate this probability if the host acts optimal.

Input

The first line contains the number of states n ($2 \le n \le 1000$), the number of rounds in the game k ($1 \le k \le 100$) and the index of the state needed for spectacular ending s ($1 \le s \le n$).

Next n lines contain transition rules for each of the states. The first n numbers of the i-th row determine the number of dice faces F_{ij} ($0 \le F_{ij} \le 1000$) which correspond to transitions to each of n states from the i-th state. The total number of dice faces satisfies $2 \le \sum_j F_{ij} \le 1000$. Than follows the denominator Q_i ($\sum_j F_{ij} \le Q_i \le 10^6$) of probabilities of all dice faces. And further along $\sum_j F_{ij}$ numerators P_{ij} ($1 \le P_{ij} \le Q_i$) of probabilities of all dice faces. The sum of probabilities of all faces equals to 1.

Output

Print maximum probability to get the spectacular ending if the host acts in an optimal way with absolute error less than 10^{-9}.

Examples

standard input	standard output
2 1 2 1 1 3 1 2 1 1 3 2 1	0.666666666667
3 3 3 1 1 0 3 2 1 0 1 1 3 2 1 1 0 1 3 2 1	0.592592592593
2 2 2 2 1 4 1 2 1 2 1 4 1 1 2	0.5

Problem K. Efficient Interception

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

The Kingdom of Berland is great and glorious, but it is currently ruled by a powerful and peevish king Vlad who got old and now sees treasons and plots all around him. He now suspects his two sons in planning a coup, so he decided to establish his control over all their communications.

The Kingdom of Berland is formed by n cities connected by m bidirectional roads in a way that makes it possible to get from any city to any other city using these roads only. The elder son of King Vlad lives in city 1, while the younger son lives in city n.

In order to control all messengers who run between city 1 and city n Vlad wants to pick some cities and establish a secret police department in each of them. It turns out that no matter how powerful you are, you are not able to keep all the things secret. If city v has a police department established in it, the mayor of this city somehow becomes aware of this. Moreover, the mayor of any city that is connected to v by a direct road becomes aware of this police departments as well.

Of course, the King wants to keep his sons unaware of his new secret police. Therefore he has three limitations.

1. There should be no way to get from city 1 to city n using roads without visiting any city that has a police department in it.
2. There can be no secret police department in city 1 and city n. Though, there can be a secret police department in a city that is connected to city 1 or city n by a direct road.
3. The total number of city mayors that are aware of at least one secret police department should be minimum possible.

You don't want to help King Vlad in his malicious games, but the problem itself looks nice so you want to find the optimal answer just for fun.

Input

The first line of the input contains a single integer t ($1 \le t \le 100$) — the number of test cases.

Then follow t test cases' descriptions. Each test case description starts with two integers n and m ($3 \le n \le 300$, $2 \le m \le \frac{n \cdot (n-1)}{2} - 1$) — the number of cities and the number of roads in the kingdom of Berland. Each of the following m lines contains two integers u_i and v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$), providing the indices of the cities connected by the i-th bidirectional road.

It is guaranteed that the sum of values of n in all t test cases doesn't exceed 300.

For each test it is guaranteed that each road connects two distinct cities, no two roads connect the same pair of cities and no road connects city 1 with city n.

Output

For each of t tests print two lines. The first line of the output should contain two integers a and k, where a is the answer, i.e. the minimum possible number of city mayors to be aware about at least one secret police department, and k is the number of secret police departments to be establish in order to achieve answer a. The next line should contain k distinct indices $x_1, x_2, \ldots, x_k$ ($1 < x_i < n$), providing where exactly to place these secret police departments in the optimal answer. If there are several optimal answers, print any of them.

Example

standard input	standard output
3 3 2 1 2 2 3 4 4 1 2 2 4 1 3 3 4 13 18 1 2 1 3 1 4 2 5 3 5 3 6 4 6 5 7 6 7 7 8 7 9 8 12 8 11 9 11 9 10 12 13 11 13 10 13	3 1 2 4 2 2 3 5 1 7

Problem L. FoodSberry

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

With the speed of light are food technologies conquering cities and towns of Berland, and, especially, its gorgeous capital City N.

FoodSberry is the largest food delivery that operates in City N. Consider the city as a two-dimensional square with sides parallel to coordinate axis. The bottom left corner of this square has coordinates $(-10^4, -10^4)$ and the top right corner has coordinates $(10^4, 10^4)$. All coordinates are given in meters.

FoodSberry has one Super Large Warehouse located at $(0, 0)$ and n dark stores (smaller warehouses), the i-th of them is located at point (x_i, y_i). Each dark store has two types of delivery: by foot and by car. Delivery by foot can be used for all points that are located no more than a meters from this dark store, while delivery by car can be used for up to b meters distances ($a \le b$, obviously). During one day each dark store is able to execute no more than c deliveries. Moreover, due to the limitation of vehicles, no more than d of these deliveries can be done by car ($d \le c$).

Super Large Warehouse is located at $(0, 0)$ and is capable of performing any number of deliveries to any locations in the city. However, using this option is expensive and FoodSberry tries to avoid it to for as long as possible.

In this problem we are going to analyze one day of FoodSberry in City N. There were m orders during that day, they are given in the order they were made and processed. The i-th order asks for the delivery to point (x'_i, y'_i). For the purpose of this problem we introduce some bold assumptions.

1. First all orders were received. Then all deliveries were executed.
2. No two orders appear in the same moment of time.
3. After each order appears, there is enough time to re-process everything. Scheduling center assumes there will be no more orders today and makes a distribution of orders between dark stores (and the warehouse). Moreover, for each delivery it is determined whether it should be done by foot or by car. This distribution is completely re-done after any new order appears, it doesn't need to depend on the previous distribution in any way.

Of course, each distribution is done with respect to limitations on distance and quantity of orders per dark store given by parameters a, b, c and d. If there is a distribution of orders that satisfies all the requirements and executes all the deliveries without using Super Large Warehouse, the scheduling program goes for it.

Assuming all distributions are made optimal, what is the minimum order index i such that there will be no way to make a valid distribution without using Super Large Warehouse?

Input

The first line of the input contains six integers n, m, a, b, c and d ($1 \le n, m \le 500$, $1 \le a \le b \le 30\,000$, $1 \le d \le c \le 1000$). Here n is the number of dark stores in City N, m is the number of orders to process, a is the maximum possible distance for delivery from dark store by foot, b is the maximum possible distance for delivery from dark store by car, c is the maximum possible total number of deliveries a dark store can execute a day, and d is the maximum possible number of deliveries by car a dark store can execute a day.

Then follow n lines containing two integers x_i and y_i ($-10\,000 \le x_i, y_i \le 10\,000$) each — coordinates of dark stores.

Finally go m lines with coordinates of orders. Each of them contains two integers x'_i and y'_i ($-10\,000 \le x'_i, y'_i \le 10\,000$).

Any points of the input can coincide, i.e. there can be two or more dark stores in the same point, two or more orders in the same point, a dark store and an order in the same point, a dark store or an order at point $(0, 0)$ and so on.

It is possible that some order is located out of reach of any dark store. Note that it is still possible to execute all the orders as FoodSberry has Super Large Warehous that is capable of doing any number of deliveries to any location in City N.

Output

If it is possible to obey all the requirements and satisfy all m orders without using Super Large Warehous, print `-1` in the only line of the output.

Otherwise, print minimum possible index i such that Super Large Warehouse is required to satisfy first i orders.

Examples

standard input	standard output
1 3 1 3 2 1 1 1 2 1 2 2 1 2	3
3 6 1 1 2 2 0 1 -2 1 2 1 -1 1 1 1 0 2 0 0 -2 1 2 1	-1

Problem M. Fine Trip

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

There are n crossings in Bytetown, connected with m bidirectional roads. The i-th road connects crossings u_i and v_i, has length of l_i meters and a *cost factor* of c_i.

In order to ease the road traffic, a special fine system is used in Bytetown. You can drive with any speed and change the speed whenever you want. However, if the maximum speed during your travel along the road i is v meters per second, you should pay $v \cdot c_i$ dollars fine.

You live near the crossing 1 and want to get to the crossing n in no more than T seconds. What is the smallest possible fine you will have to pay if you choose the route and the speed for each road optimally?

Input

In the first line of the input there are three numbers n, m and T ($1 \le n \le 2000$, $1 \le m \le 100\,000$, $1 \le T \le 10^9$).

The i-th of the next m lines contains four numbers u_i, v_i, l_i, c_i, describing the i-th edge according to the statement ($1 \le u_i, v_i \le n$, $u_i \ne v_i$, $1 \le l_i, c_i \le 10^9$).

It is guaranteed that it is possible to reach the crossing n starting at the crossing 1.

Output

Print single real number — the smallest possible fine. Your answer will be considered correct if its relative or absolute error doesn't exceed 10^{-6}.

Examples

standard input	standard output
3 3 100 1 3 100 100 1 2 100 24 2 3 100 24	96.0000000000
3 2 10 1 2 9 1 2 3 1 1000	119.8736659610
4 5 9 1 2 6 7 1 3 6 2 2 3 8 7 2 4 3 4 3 4 1 7	4.1478114200

Problem N. Integer Perimeter

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	512 megabytes

You are given six positive integers a, b, c, d, e and f. You need to find out whether there exists a triangle ABC with the following properties.

- $AB/BC = a/b$;
- $BC/CA = c/d$;
- $CA/AB = e/f$;
- Area of ABC is positive;
- The perimeter of the triangle is integer.

If such triangle exists, find the minimum possible value of its perimeter.

Input

The input consists of six lines, each containg one integer a, b, c, d, e and f ($1 \le a, b, c, d, e, f \le 1000$) respectively.

Output

If there is no triangle satisfying all the properties, print `-1`. Otherwise, print one integer — the minimum possible perimeter of such triangle.

Examples

standard input	standard output
1 1 2 2 3 3	1
1 2 3 4 5 6	-1
1 2 2 3 3 1	-1

Problem O. Treeshop

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	512 megabytes

Max enjoys to play chess a lot. The only thing in this world that excites him more is to come up with ideas for programming contest's problems. This particular problem is of an ultimate delight for him as he was able to join his interests in one masterpiece.

Max has invented a tree-coordinate two-dimensional board and you are going to move a chip named treeshop along it. Formally, you are given two connected undirected acyclic graphs (also known as trees) of size n_1 and n_2 respectively. The position of a treeshop is defined by a pair of vertices (u, v), where u is some vertex of the first tree and v is some vertex of the second tree.

The distance between two vertices of the same tree equals to the number of edges the only simple path connecting these two vertices consists of. A treeshop is able to move from position (u, v) to position (u', v') in one move if and only if the distance between vertices u and u' equals to the distance between vertices v and v'. Obviously, if it is possible to get from (u, v) to (u', v') in one move, it is also possible to get from (u', v') to (u, v) in one move.

You are given q pairs of positions. For each of them you need to compute the minimum number of moves it will take a treeshop to get from one of these positions to the other.

Input

The first line of the input contains integers n_1, n_2 and q ($1 \le n_1, n_2, q \le 200\,000$) — the number of vertices in the first tree, the number of vertices in the second tree and the number of pairs of positions to process.

The following $n_1 - 1$ lines describe edges of the first tree. Each of them contains two integers u_i and v_i ($1 \le u_i, v_i \le n_1$, $u_i \ne v_i$) — the indices of vertices connected by the corresponding edge.

The following $n_2 - 1$ lines describe the second tree in exactly the same way.

Then follow q lines with the description of all pairs of positions to process. The i-th of these lines contains four integers s_1, s_2, t_1 and t_2 ($1 \le s_1, t_1 \le n_1$, $1 \le s_2, t_2 \le n_2$) — first two describe the initial position and the next two describe the target position.

Output

Print one integer for each query. If it is possible to get from the initial position to the target position by doing treeshop moves, print the minimum number of moves required to do so. Otherwise, print `-1`.

Example

standard input	standard output
4 5 7 1 2 2 3 2 4 1 2 2 3 3 4 4 5 1 1 2 5 1 5 4 1 1 5 4 2 2 5 2 3 2 1 2 5 3 2 3 2 4 4 1 2	-1 2 -1 2 3 0 1