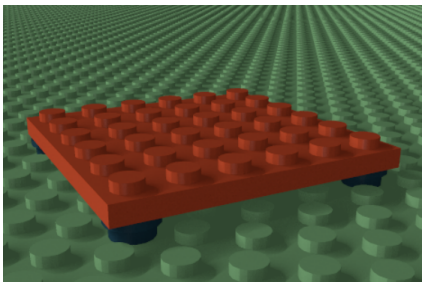
Problem A. Alternative Architecture

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



Placing a 6×6 plate diagonally.

In his free time, Thomas greatly enjoys working on the extensive Lego project that he has built in his attic, adding house after house to his miniature city. However, he has become a bit bored with the completely rectangular layout that is enforced by the little studs of the huge base plate that his city is built on.

After an exchange with some other Lego creators he came across a technique that will allow him to place his buildings at different angles. Each building rests on a rectangular ground plate, to the underside of which he attaches four round 1×1-plates in the corners. These 1×1-plates are then placed on four studs of the base plate, like in Figure.

If the ground plate of the building is $a \times b$ studs, what is the number of orientations it can be placed in using this technique, so that all the corner plates exactly fit on studs of the base plate?

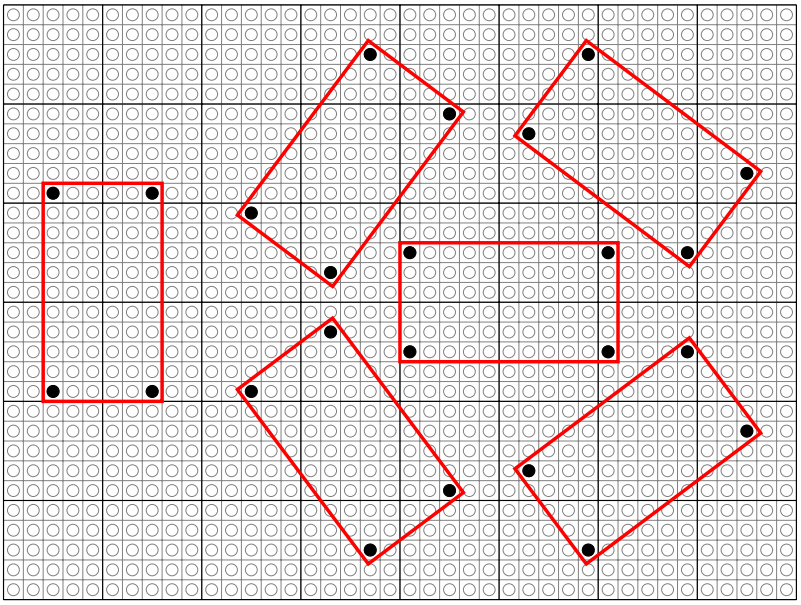


Illustration of the first example case. There are many ways of placing a 6×11 plate, but only 6 distinct orientations.

Input

The input consists of:

- One line with two integers a and b ($2 \leq a, b \leq 10^6$), the dimensions of the ground plate the building is resting on.

Output

Output one integer, the number of different orientations the ground plate can be placed in.

Example

Input	Output
6 11	6
26 26	5
123 456	2
3 3	1

Problem B. Breeding Bugs

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

A single cicada. Picture by parlansky, Pixabay

This year is a good year for North America. 2022 is one of the few years where no brood of the periodical cicada is hatching and thus, no swarms will destroy the crops on the fields.

Those periodical cicadas have a somewhat strange property: They have highly synchronized life cycles, which means that almost all individuals in a local population emerge in the same year, resulting in periodical cicada plagues. Even odder is the fact that the periodicities of those life cycles appear to be prime, for example 13 or 17 years. The best theory for this so far is that a prime periodicity lets them avoid predators with shorter population cycles since a brood emergence of cicadas will rarely coincide with a predator's population boost.

But nobody likes cicada plagues, so this prime periodicity is now your problem. Your hope is that cicadas with non-prime periodicity will not be able to avoid predators anymore and that there will be fewer cicada plagues as a result. So, to prevent the next plague, you forge a plan to breed different cicada types to get a new type with non-prime periodicity. If you mate a cicada of a type with periodicity a with another cicada of a type with periodicity b, you assume to get a cicada of a type with periodicity $a + b$. You have already captured n cicadas to breed but you don't know which will mate. Therefore, you decided to set some cicadas free such that the remaining ones can mate this year in any way they want without producing a cicada of a type with prime periodicity. How many of your cicadas can you keep at most?

Input

The input consists of:

- One line with a single integer n ($1 \leq n < 750$), the number of cicadas.
- One line with n integers $p_1, \ldots, p_n$ ($1 \leq p_i < 10^7$), where p_i denotes the periodicity of the ith cicada.

Output

Output a single integer, the maximum number of cicadas you can keep.

Example

Input	Output
8 1 2 3 4 5 6 7 8	4
5 7 13 2 2 4	4

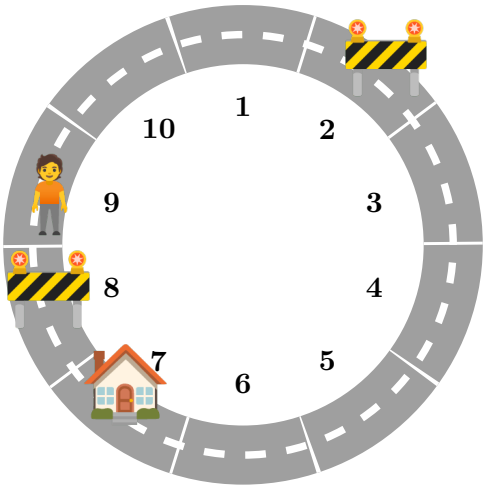
Problem C. Chaotic Construction

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

The city Gircle has only one street, and that street is cyclic. This was very convenient in times when people didn't carry a device with compass, GPS and detailed maps around in their pockets, because you only have to walk in one direction and will certainly arrive at your destination. Since Gircle's founding a lot of time has passed. Civil engineers now know a lot more about road network design and most people have immediate access to reliable and accurate navigation systems. However, the passage of time also affected the old street surface and more and more cracks and potholes appeared.

The local government has finally decided to improve the situation, but preserving the city's historic appeal and building new streets are unfortunately mutually exclusive. Because tourism is vital for Gircle's economy, the government's only viable option for improving the situation is to renovate segments of the street when necessary. Gircle's street is very narrow, so a construction site at a street segment makes it impossible for citizens to pass that segment or even leave or enter it.

As a member of the *Gircle Construction and Planning Commission* (GCPC), you always know when one of the n street segments is closed or reopened. Naturally, the citizens expect you to tell them whether the trips they want to do are currently possible.



Depiction of the query "`? 9 7`" in the example input.

Input

The input consists of:

- One line with two integers n ($2 \leq n \leq 10^5$) and q ($1 \leq q \leq 10^5$), the number of street segments and the number of events. No street segment is initially closed.
- q lines, each describing an event. Each event is described in one of the following ways:
  - "`- a`": Segment a ($1 \leq a \leq n$) is closed. It is guaranteed that segment a was open before.
  - "`+ a`": Segment a ($1 \leq a \leq n$) is reopened. It is guaranteed that segment a was closed before.
  - "`? a b`": A person asks you if it is possible to go from segment a to segment b ($1 \leq a, b \leq n$ and $a \neq b$).

Output

For each event of the form "`? a b`", print one line containing the word "`possible`", if it is possible to move from segment a to segment b, or "`impossible`" otherwise. If a or b are currently closed, the answer is "`impossible`".

Example

Input	Output
10 12 ? 1 5 - 2 - 8 ? 9 2 ? 9 8 ? 9 7 ? 6 7 ? 3 7 ? 1 9 ? 9 1 + 8 ? 10 3	possible impossible impossible impossible possible possible possible possible possible

Use fast I/O methods

Problem D. Dynamic Collection

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

You are given a collection of elements, each of which is a positive integer. Duplicate elements are allowed in this collection. There are two types of operations that you can perform on this collection. Operation 1 is for updating the collection, and Operation 2 is for querying the collection. The format for these operations are as follows:

- Operation 1: "1 k"
- Operation 2: "2 a b"

For Operation 1, k is an integer upon which the following rules will be applied in order:

- If the element k is already in the collection, the collection remains the same.
- If the element k is greater than any of the elements in the collection, it is added to the collection.
- If the element k is not in the collection then the first occurrence of the smallest element greater than k is replaced by k.

For Operation 2, a and b are two integers that define a range. The purpose of this operation is to query how many elements in the collection fall within this range (inclusive).

Input

The input consist of an unique test case.

The first line contains two positive integers, n and q ($1 \leq n,\ q \leq 10^6$) which represent the number of elements in the collection and the total number of operations to be performed on the collection, respectively.

The second line contains exactly n space-separated integers a_1, a_2, ..., a_N ($1 \leq a_i \leq 10^9$, $i = 1, 2, \ldots, N$).

The next q lines each represent an operation. Each operation is either of type 1 or type 2, and follows the format: "1 k" or "2 a b", ($1 \leq a,\ b,\ k \leq 10^9$; $a \leq b$).

Output

For each Operation 2 (query operation) in the input, print a single line containing a positive integer, which represents the total number of elements in the collection that fall within the range $[a,\ b]$.

Example

Input	Output
10 11 7 1 7 1 3 9 7 9 10 4 2 2 8 1 8 2 2 8 2 1 20 1 20 2 1 20 2 7 12 1 5 2 7 12 1 12 2 7 12	5 6 10 11 6 5 6

Use fast I/O methods

Explanation

Let's consider the first four operations in the example:

The initial collection is the following:

$$C = [7,\ 1,\ 7,\ 1,\ 3,\ 9,\ 7,\ 9,\ 10,\ 4]$$

after the operation "2 2 8" the output is 5.

After the operation "1 8" the collection becomes:

$$C = [7,\ 1,\ 7,\ 1,\ 3,\ 8,\ 7,\ 9,\ 10,\ 4]$$

Now, after operation "2 2 8" the output is 6.

After the operation "1 20" the collection becomes:

$$C = [7,\ 1,\ 7,\ 1,\ 3,\ 8,\ 7,\ 9,\ 10,\ 4,\ 20]$$

Problem E. Enjoyable Entree

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

For many days now, the canteen in Hilbert's Hotel has been offering its famous *Fibonacci Soup*. Today is the nth day that they offer the soup, and the recipe changes every day:

- On the first day, it is a π*-tato soup*, made from a blend of local potato varieties, and served with onion strips and celery.
- On the second day, it is a τ*-mato soup*, consisting of puréed tomatoes, together with carrots, onions, garlic, and basil leaves on top.
- On every day after that, the soup is made from two ingredients, the soup of the previous day and the soup of the day before that, mixed together in equal parts.

π-tato and τ-mato soup. Picture by RitaE, Pixabay

Find the composition of today's Fibonacci Soup.

Input

The input consists of:

- One line with a single integer n ($1 \leq n \leq 10^{18}$), the current day.

Output

Output two real numbers π and τ ($0 \leq \pi, \tau \leq 100$), giving the percentages of π-tato soup and τ-mato soup in the nth day's Fibonacci soup. Your answer will be accepted if the absolute error is at most 10^{-6}.

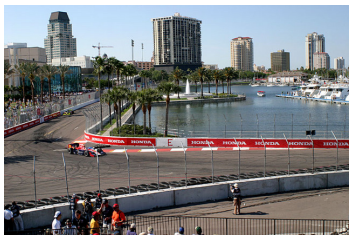
Example

Input	Output
1	100.0 0.0
3	50.0 50.0
7	34.375 65.625

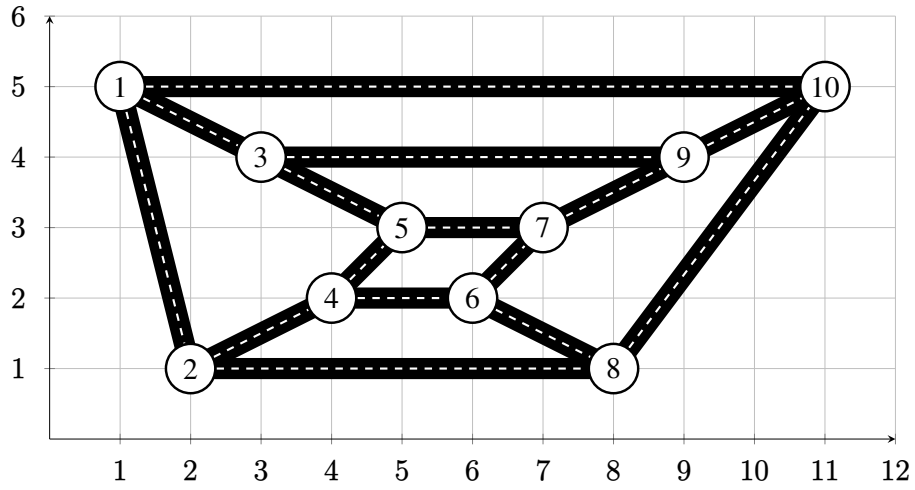
Problem F. Formula Flatland

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

Flatland is happy to announce that this year – for the first time ever – the Formula One comes to Flatland to arrange the Grand Prix of Flatland. As with many other cities, Flatland is not able to build a dedicated circuit for the race. Therefore, Flatland decided to close off some of its normal streets and crossings to form a circuit. After your excellent work as an organizer of last year's *Flatland Olympics*, you were hired to find a suitable circuit. Since closed off streets are annoying to the people who live there, you would like to minimize the number of crossings that need to be closed off for the race.



Part of a street circuit. Photo by Ronnie Boone, wikipedia



Visualization of the second example. One possible optimal circuit would be: $(4, 5, 7, 6)$.

Your job only consists of selecting some road segments which form a circle but are connected by as few crossings as possible. Note that even though all roads in Flatland are bidirectional, they can only be used in one direction during the race for safety reasons.

Input

The input consists of:

- One line with two integers n and m ($4 \leq n \leq 10^5$ and $5 \leq m \leq 3 \cdot 10^5$), the number of crossings and the number of road segments in Flatland.
- n lines, each with two integers x and y ($0 \leq x, y \leq 10^9$), the ith line describes the position of the ith crossing on a map of Flatland. No two crossings are at the same position.
- m lines, each with two integers a and b ($1 \leq a, b \leq n$ and $a \neq b$), describing that the ath and bth crossing are connected by a road segment. Two crossings are connected by at most one road segment.

It is guaranteed that two road segments only intersect in a crossing they both start or end at. Further, it is guaranteed that each crossing on the map corresponds to an actual crossing in the sense that at least three road segments intersect.

Output

Output a single integer, the minimum number of crossings the racetrack must contain.

Example

Input	Output
4 6 0 0 3 0 0 3 1 1 1 2 1 3 1 4 2 3 2 4 3 4	3
10 15 1 5 2 1 3 4 4 2 5 3 6 2 7 3 8 1 9 4 11 5 1 2 1 3 1 10 2 4 3 5 4 5 4 6 5 7 6 7 6 8 7 9 8 10 9 10 2 8 3 9	4

Use fast I/O methods

Problem G. Guessing Game

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

A plentiful selection of delicious vegetables.
Picture by Alexandr Podvalny, Unsplash

Every year, the top gardeners of the cities Greenville and Tomatown compete against each other in the *Grand Gardening Competition*. The competition consists of some number of examinations which take place over the course of one week from Monday to Sunday. In each examination, one gardener from Greenville and one gardener from Tomatown present their products to a neutral jury. A few days in advance, both gardeners officially announce how many products of each type of vegetable, fruit or berry they plan to present. During the examination, the jury then evaluates the size, weight, diversity, beauty and taste of the presented products. After careful consideration, the jury finally declares one of the two competing gardeners to be the winner of the examination.

Alan and his friends are all enthusiastic gardeners, but since they do not live in Greenville or Tomatown, they can not submit their own vegetables to the competition. However, they have started their own private contest, where they try to predict the results of the single examinations. In this contest, each participant is allowed to pick one examination from each of the seven competition days and predict its winner. If this prediction turns out to be correct, the participant is awarded one point. To keep their guessing game interesting, Alan and his friends agreed that a prediction for an examination can not be handed in after the competing gardeners have announced which products they are going to present.

By using his connections to the gardening scenes of Greenville and Tomatown, Alan has consistently managed to score more points than all of his friends in the previous years. However, when he woke up this year on Monday, the first day of the competition, Alan realized that he had completely forgotten to submit his predictions! Of course, he immediately sprinted towards his computer and tried to submit his bets. Unfortunately, all gardeners which were scheduled to present their products between Monday and Friday had already announced their selections, so Alan could only submit his predictions for two examinations on Saturday and Sunday. He then hastily grabbed the competition schedule and started to compare the announced examinations to the predictions made by him and his friends.

Help Alan to determine whether there still remains a tiny chance that he can once more win the Gardening Competition Prediction Contest.

Input

The input consists of:

- One line with a single integer n ($1 \leq n \leq 5 \cdot 10^4$), the number of Alan's friends.
- One line with seven positive integers $d_1, \ldots, d_7$ ($d_1 + \cdots + d_7 \leq 10^5$), indicating that exactly d_i examinations will take place on the ith competition day.
- n lines, each describing the predictions of one of Alan's friends. Each line consists of seven integers $b_1, \ldots, b_7$ ($1 \leq |b_i| \leq d_i$). If b_i is positive, the $|b_i|$-th examination on day i is predicted to be won by the gardener from Greenville. If it is negative, the gardener from Tomatown is predicted to win the examination.
- One line with two non-zero integers b_6, b_7 ($1 \leq |b_i| \leq d_i$), encoding Alan's predictions for Saturday and Sunday in the same manner as the predictions of his friends.

Output

If it is possible for Alan to score more points than any of his friends, output `possible`. Otherwise, output `impossible`.

Example

Input	Output
3 4 4 4 4 4 4 4 1 1 1 1 4 -2 1 2 2 2 2 -4 1 -1 3 3 3 3 -3 3 3 -2 -1	impossible
3 4 4 4 4 4 4 4 4 3 2 1 4 1 1 2 4 4 2 2 4 2 2 3 3 4 1 3 2 -2 -1	possible

Problem H. GCD Harmony

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

Consider a tree with undirected edges, where each node has an integer value. Adjacent nodes are said to be *GCD-harmonic* if the greatest common divisor (GCD) of their values is strictly greater than 1.

You can modify the value of any tree node to any positive integer. The cost of this operation is equal to the new node value, regardless of the node's original value. You can change as many node values as needed, and node values do not need to be unique.

What is the minimum total cost to make every pair of adjacent nodes in the tree GCD-harmonic?

Input

The first line of input contains a single integer n ($2 \leq n \leq 5000$), which is the number of nodes in the tree. Tree nodes are numbered from 1 to n.

Each of the next n lines contains an integer v ($1 \leq v \leq 100$). These are the initial values of the nodes (which are not guaranteed to be unique), in node number order.

Each of the next $n - 1$ lines contains two integers a and b ($1 \leq a,\ b \leq n$, $a \neq b$), indicating a tree edge between nodes a and b. It is guaranteed that these edges form a tree.

Output

Output a single integer, which is the minimum total cost to make every pair of adjacent nodes in the tree GCD-harmonic.

Example

Input	Output
6 5 6 3 4 9 12 1 2 1 3 1 4 1 6 3 5	6
3 1 2 3 3 1 2 3	4

Problem I. Improving IT

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

The Summit supercomputer cost 200 million dollar to build.

Your best friend is part of the business team at the *Global Center for Parallel Computing* (GCPC). She is responsible for buying and selling the hardware that is powering the system that will be in use for the next n months. Currently, she is planning the CPU replacement cycle for a single CPU. To ensure that the system is always up-to-date, the CPU must be replaced at least every m months. Fortunately, she can sell the replaced CPU to lower the overall costs to operate the new system. However, storage capacity is pricey, and she has to accept the resale value the CPU has in the month it is replaced. That means, when a CPU that was used for j months is replaced in month i, you need to sell the current CPU for the value it has after j months of usage and buy a new CPU for the price of the ith month. She already compiled a list of CPU prices for the next n months including their resale value after 1 to m months. Note that you definitely need to buy a CPU in month 1 and you need to sell the last CPU in month $n+1$. How much money does the system cost at least over the n months?

Input

The input consists of:

- One line with two integers n and m ($1 \leq n, m$ and $n \cdot m \leq 5 \cdot 10^5$).
- n lines; the ith line has an integer c ($0 \leq c \leq 10^9$), the cost of a CPU in month i, followed by $\min(m, n-i+1)$ integers c_j ($0 \leq c_j \leq 10^9$), the money you earn by selling this CPU after $j > 0$ months.

Output

Output a single integer, the minimum total cost. Note that this number can be negative if reselling CPUs was profitable.

Example

Input	Output
4 3 1000 900 800 900 700 600 500 400 1200 1200 1300 600 500	100
3 2 200 300 400 400 300 200 300 500	-400

Problem J. Jesting Jabberwocky

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

The famous card game manufacturer *Greatest Cards Production Company* (GCPC) has just created the brand new card game *Jabberwocky*. In this game, everyone gets the same amount of cards – which might be quite a lot – and each card belongs to one of four different suits: hearts, diamonds, clubs, or spades.

In Example 1, Alice has to move at least two cards to sort her hand.

As huge card game nerds, Alice and her friends are very hyped about meeting up and trying out the card game everybody seems to talk about these days. Due to a traffic jam, Alice is a bit late to the party and her friends are impatiently waiting for her. They have already distributed all cards and everybody is ready to go, except for Alice. She has just picked up her cards and insists on sorting them by suit first. For that, she repeatedly picks one card from her hand and inserts it somewhere else until her cards are grouped by suit. Her friends are getting increasingly annoyed with Alice and she wants to sort her cards as quickly as possible. How many cards does Alice need to move before they can start playing?

Input

The input consists of:

- One line with a string s ($1 \leq |s| \leq 10^5$), representing the suits of Alice's cards as they are initially ordered. The string consists of the characters `h`, `d`, `c`, and `s` (hearts, diamonds, clubs and spades).

Output

Output a single integer, the minimum number of cards Alice has to move in order to sort the cards by suit.

Example

Input	Output
`hccdhcd`	2
`cchhdshcdshdcsh`	7

Problem K. K.O. Kids

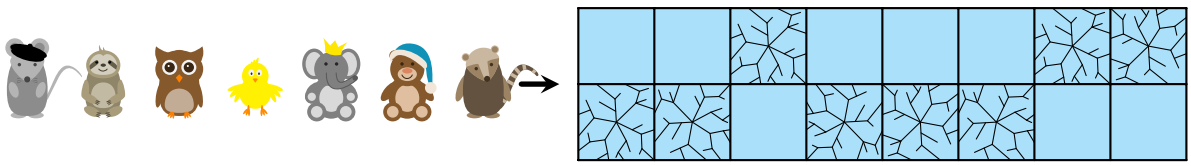
Source file name:	Kids.c, Kids.cpp, Kids.java, Kids.py
Input:	`Standard`
Output:	`Standard`

On his birthday party, Glen wants to play the most exciting game. It is called *Splash Game*. For this, his parents built a bridge, which goes over the full length of the family pool and can be seen as a $2 \times n$ grid: it consists of n steps and at each step, there are two plates. The players go over the bridge one after the other in a fixed order. At each of the n steps, one of the two plates is fake and the player will fall into the pool with a big *Splaaaaash* when she steps onto it.

Of course, a participant can be lucky and guess the real plate and will not fall (she might still fall later). Also the first player really has a tough time. To make it to the other side, she would need to guess the real plate at every step. The later players have the advantage that they can see what the others are doing and hence know for the already entered steps, which plate is the real one (if a player guesses the real plate, everybody sees it; if she guesses the fake one and falls, everybody knows that the other plate is the real one).

The players proceed by a simple strategy. The first player starts by choosing the left plate on the first step. If she is correct, she switches to the right side and she will keep switching the side at every step (it is common knowledge that switching is a good idea). Every other player, once it is her turn, follows the correct choices as far as they are known and, afterwards, applies the switching strategy as well, i. e., if she stepped on the left plate on the previous step, she now steps on the right one and vice versa.

Of course, the game is only fun if at least a few kids make it to the other side of the bridge. But it shouldn't be too many either, since everybody has a great laugh when somebody is falling into the water. Given the number of kids and the planned layout of fake and real plates, output how many kids make it to the other side of the bridge.



The bridge layout for Example Input 4 (cracked squares indicate the fake plates). The first player will guess the first step correctly, but fall on the second step. The second player thus knows the correct choices for the first two steps and guesses the third and fourth one correctly by switching. In the end, three of the seven kids make it to the other side.

Input

The input consists of:

- One line with integers n, k ($1 \leq n, k \leq 10^3$), the length of the bridge and the number of kids.
- One string s of length n consisting of characters `L` and `R`. An `L` on position i indicates that the real plate at step i is the left one, an `R` indicates the right plate is the real one.

Output

Output a single integer, the number of kids who make it to the other side of the bridge.

Example

Input	Output
3 5 LRL	5
3 2 RRR	0
3 5 LLL	3
8 7 LLRLLLRR	3

Problem L. Leaderboard Effect

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

You are in charge of a programming contest with a number of distinct problems and an overall time limit. The judges provide for each problem a reading time, a coding time, and a solution probability.

During the contest, teams can view a leaderboard that shows the accepted submissions for each team. Seeing the leaderboard can have an effect on teams' strategies! All teams will follow the following strategy when solving problems in the contest:

1. Choose to read a problem. To choose which problem to read, they first gather all the problems they haven't yet read (if they have read all problems, they will spend the rest of the contest doing nothing). If none of these problems have any solutions, they will choose to read one uniformly at random. Otherwise, they will choose a problem to read, randomly weighted by the number of solutions that it has. For example, if there are three problems A, B, C with 3, 1, and 0 solutions respectively, they will choose to read problem A with probability $\frac{3}{4}$, problem B with probability $\frac{1}{4}$, and problem C with probability $\frac{0}{4}$.
2. After choosing any given problem, they first read it. This always takes the given reading time for that problem. After they finish reading the problem, they will know if they can solve it or not. The team will be able to solve the problem with the given probability (and this takes no time to decide). If they can't solve the problem, the team goes back to step 1 and chooses another problem to read. Otherwise, they spend the given time coding the problem (even if there is not enough time left in the contest to finish) before going back to step 1 to choose another problem to read. Once a team finishes coding the chosen problem, the problem is solved (receiving judgement takes no time); each problem solution affects the leaderboard, and therefore the problems other teams will attempt.

At all moments of time, the leaderboard will first update with all the solutions that happened at that time, then teams will see the updated leaderboard.

Let $f(m,\ i)$ be the expected number of teams that will solve problem i if there are m teams participating in the contest. Let $g(i) = \lim_{m\to\infty} \left[\frac{f(m,\ i)}{m}\right]$, which is the expected proportion of teams that will solve problem i for a large enough number of teams. Output $g(i)$ for all problems i in the set.

Input

The first line of input contains two integers n ($1 \leq n \leq 17$) and t ($1 \leq t \leq 100$), where n is the number of problems in the contest, and t is the time limit in minutes.

Each of the next n lines contains two integers r and c ($1 \leq r,\ c \leq t$) and one real number p ($0.0 \leq p \leq 1.0$). Each line describes a problem, where r is reading time in minutes, c is coding time in minutes, and p is the probability of solving the problem.

Output

Output n lines. Each line contains a single real number, which is the expected proportion of teams that will solve that particular problem, given a large number of teams. Output these proportions for the problems in the same order as the input. The answers are accepted with absolute error at most 10^{-6}.

Example

Input	Output
4 42 10 10 0.75 10 10 0.75 10 12 1 10 12 1	0.45625 0.45625 0.296875 0.296875
4 42 10 12 0.75 10 12 0.75 10 10 1 10 10 1	0.203125 0.203125 0.683238636363636 0.683238636363636
5 100 40 60 0.6 40 61 1 10 40 0.3 10 40 0.4 10 40 0.5	0.12 0 0.112628571428571 0.159739682539683 0.206444444444444

Problem M. Mirror Madness

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

Yesterday, a new attraction was opened at the local funfair: a hall of mirrors. This is a labyrinth in which all the walls are covered with mirrors so that visitors lose their sense of direction in a sea of reflections. The layout of the labyrinth can be described by a polygon with all sides parallel to either the x-axis or y-axis.

On the opening day, many visitors got so lost that the ride operators had to intervene and help them find their way out. To better understand where visitors get lost the most, the operators decided to install a monitoring system. This system involves an invisible laser beam running through the hall of mirrors at foot level, so that the movement of the visitors can be tracked by observing when and where the laser beam gets interrupted.

The laser beam starts at some boundary point of the polygon at an angle of 45 degrees to the wall. Whenever it hits a mirror, it bounces off in a 90 degree angle. To get their monitoring system to work, the operators are planning to install sensors at each of the first m bouncing points of the laser beam. Find the locations where the sensors need to be installed.

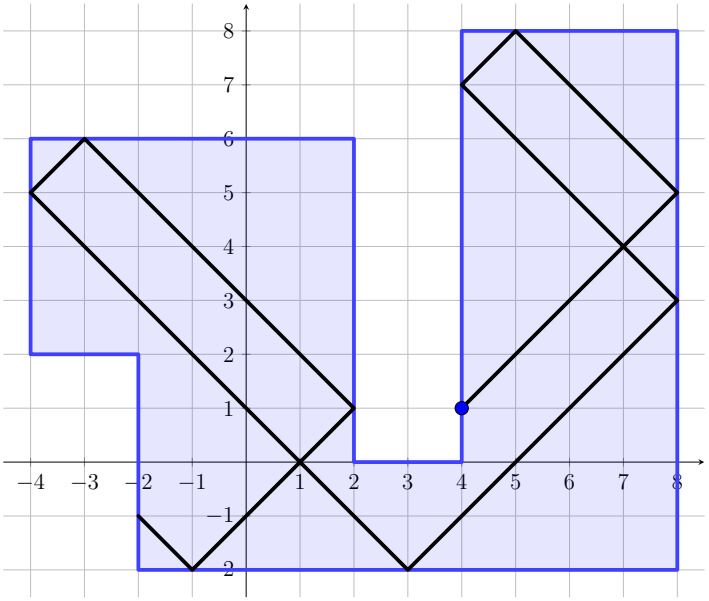


Illustration of the second Example case.

Input

The input consists of:

- One line with two integers n and m ($1 \leq n, m \leq 5 \cdot 10^5$), the number of vertices of the polygon and the number of bounces.
- n lines, each with two integers x and y giving the coordinates of one vertex.
- One line with integers x_s and y_s, the coordinates of the starting point.

Additionally, the input satisfies the following constraints:

- The vertices of the polygon are given in counterclockwise order.
- The edges of the polygon do not touch or intersect each other, except for consecutive edges, which share their endpoints.
- The edges of the polygon alternate between horizontal and vertical.
- The perimeter (the total length of all sides) of the polygon does not exceed 10^6.
- All coordinates in the input have absolute value at most 10^6.
- All coordinates of the vertices of the polygon and exactly one coordinate of the starting point are even (so the laser never hits a vertex of the polygon).
- The starting point is on the boundary of the polygon.
- The initial direction of the laser beam is $(1, 1)$, and this points inside the polygon.

Output

Output m lines, each with two integers x and y, giving the coordinates of the bounce locations in order.

Example

Input	Output
4 6 0 0 10 0 10 10 0 10 1 0	10 9 9 10 0 1 1 0 10 9 9 10
10 10 -2 -2 8 -2 8 8 4 8 4 0 2 0 2 6 -4 6 -4 2 -2 2 4 1	8 5 5 8 4 7 8 3 3 -2 -4 5 -3 6 2 1 -1 -2 -2 -1