Problem A. Alex is Right

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

While on the first leg of their journey to Beijing (China) for the 2018 ICPC World Finals, Alex and Timothy (two of the team members) were in a vigorous debate. The discussion was about their upcoming 13-hour flight from Washington, DC to Beijing. More specifically, whether the airplane would travel (a) west over the Pacific, or (b) east over the Atlantic, or (c) North then South passing by the North Pole. Alex argued that their flight would obviously fly them relatively close to the North Pole but Timothy, the self-proclaimed "Geometry Master", contested this as nonsense saying that there's no way the pilots would choose such a suboptimal route.

To Timothy's disappointment, once they landed in Washington, DC, Alex pulled up a flight map and proved that he was right by showing Timothy as well as Arup and Eric (who both supported Timothy's viewpoint) that their upcoming flight would indeed fly a route close to the North Pole.

Given the locations of two distinct points (cities) on the earth, calculate the minimum Euclidean distance between the airplane and the North Pole that will be achieved during the shortest possible flight around the surface of the earth between these two cities, and determine if Alex's claim is correct. Assume that the airplane is a point on the surface of the earth at all time, i.e., the airplane does not have an altitude.

Alex's claim can be formally defined as follows:

"The minimum Euclidean distance from the airplane to the North Pole, while traveling along an optimal (shortest) flight around the earth, is strictly less than that of the distance from the North Pole to both the starting and ending points."

The Euclidean distance between two points (x_0, y_0, z_0) and (x_1, y_1, z_1) is defined as $\sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$. Note that this is not the same as the arc distance that would need to be traveled along the surface of the earth.

To refresh your memory, below is the conversions from latitude and longitude to Cartesian coordinates on a sphere of radius R:

$$x = R \cdot \cos(latitude) \cdot \cos(longitude)$$

$$y = R \cdot \cos(latitude) \cdot \sin(longitude)$$

$$z = R \cdot \sin(latitude)$$

Notes:

- Use 6371 km for the radius of the earth in your calculations.
- The North Pole is located at 90 degrees latitude and 0 degrees longitude.

Input

The first line of input consists of a single integer, T ($1 \le T \le 500$), representing the number of trips Alex and Timothy went on. Each trip consists of two input lines. The first line of each trip consists of two space-separated real numbers latitude and longitude ($-90 \le latitude < 90$; $-180 < longitude \le 180$)

with exactly 6 digits after the decimal point, representing (respectively) the latitude and longitude of the departure city of this trip. The second input line of each trip contains the latitude and longitude of the destination city in the same format as the departure city. It is guaranteed that the departure and destination cities will be at distinct location (and not on the North Pole) and that there will be exactly one shortest possible flight path between these two cities.

Output

For each trip, output two lines. For the first line, output either Alex or Timothy representing who was right for this trip. On the next line for each trip, output the minimum Euclidean distance between the North Pole and the plane that will be achieved while flying from the departure city to the destination city. Output your answer with exactly 6 digits after the decimal point. Answers will be judged correct if the absolute error is at most $1e^{-6}$ km. Output a blank line after each trip.

Note 1: Use the value of PI (π) provided by the library of your chosen language.

Note 2: It is guaranteed that in cases where Alex is right, the absolute value of the difference between the minimum distance along the flight path to the North Pole and the minimum distance to the destination is no less than 1 km.

Example

Input	Output

2
38.946474 -77.437568
39.918620 116.443983
16.000000 175.000000
-45.000000 -93.000000
	Alex
	928.608923
	
	Timothy
	7668.327025

Problem B. Buildings

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

As a traveling salesman in a globalized world, Alan has always moved a lot. He almost never lived in the same town for more than a few years until his heart yearned for a different place. However, this newest town is his favorite yet - it is just so colorful. Alan has recently moved to Colorville, a smallish city in between some really nice mountains. Here, Alan has finally decided to settle down and build himself a home - a nice big house to call his own.

In Colorville, many people have their own houses - each painted with a distinct pattern of colors such that no two houses look the same. Every wall consists of exactly $n \times n$ squares, each painted with a given color (windows and doors are also seen as unique "colors"). The walls of the houses are arranged in the shape of a regular m-gon, with a roof on top. According to the deep traditions of Colorville, the roofs should show the unity among Colorvillians, so all roofs in Colorville have the same color.

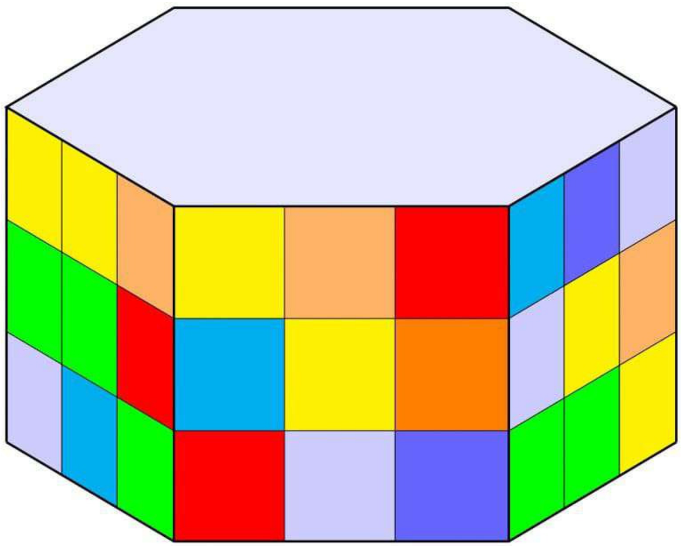

Figure B.1: **This picture** Example house design for $n = 3$, $m = 6$.

Of course, Alan wants to follow this custom to make sure he fits right in. However, there are so many possible designs to choose from. Can you tell Alan how many possible house designs there are? (Two house designs are obviously the same if they can be translated into each other just by rotation.)

Input

The input consists of:

- one line with three integers n, m, and c, where
  - n ($1 \leq n \leq 500$) is the side length of every wall, i.e. every wall consists of $n \times n$ squares;
  - m ($3 \leq m \leq 500$) is the number of corners of the regular polygon;
  - c ($1 \leq c \leq 500$) the number of different colors.

Output

Output s where s is the number of possible different house designs. Since s can be very large, output s mod $(10^9 + 7)$.

Example

Input	Output
1 3 1	1
2 5 2	209728

Problem C. Joyride

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

It is another wonderful sunny day in July – and you decided to spend your day together with your little daughter Joy. Since she really likes the fairy-park in the next town, you decided to go there for the day. Your wife (unfortunately she has to work) agreed to drive you to the park and pick you up again. Alas, she is very picky about being on time, so she told you exactly when she will be at the park's front entrance to pick you up and you have to be there at exactly that time. You clearly also don't want to wait outside, since this would make your little daughter sad she could have spent more time in the park!

Now you have to plan your stay at the park. You know when you will arrive and when you will have to depart. The park consists of several rides, interconnected by small pavements. The entry into the park is free, but you have to pay for every use of every ride in the park. Since it is Joy's favorite park, you already know how long using each ride takes and how much each ride costs. When walking through the park, you obviously must not skip a ride when walking along it (even if Joy has already used it), or else Joy would be very sad. Since Joy likes the park very much, she will gladly use rides more than once. Walking between two rides takes a given amount of time

Since you are a provident parent you want to spend as little as possible when being at the park. Can you compute how much is absolutely necessary?

Input

The input consists of

- one line with an integer x ($1 \leq x \leq 1000$) denoting the time between your arrival and the time you will be picked up (in minutes);
- one line with with three integers n, m, and t, where
  - n ($1 \leq n \leq 1000$) is the number of rides in the park;
  - m ($1 \leq m \leq 1000$) is the number of pavements;
  - t ($1 \leq t \leq 1000$) is the number of minutes needed to pass over a pavement from one ride to another.
- m lines each containing two integers a and b ($1 \leq a, b \leq n$) stating that there is a pavement between the rides a and b.
- n lines each containing two integers t and p ($1 \leq t, p \leq 10^6$) stating that the corresponding ride takes t minutes and costs p Euro.

You always start at ride 1 and have to return to ride 1 at the end of your stay, since the entry is located there. This means that you have to use the ride 1 at least twice (once upon entry and once upon exit). You can take a ride more than once, if you have arrived at it.

Output

Output one line containing either a single integer, the minimum amount necessary to stay x minutes in the park, or `It is a trap.` (including the period) if it is not possible to stay exactly x minutes.

Example

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

Problem D. Pants On Fire

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

Donald and Mike are the leaders of the free world and haven't yet (after half a year) managed to start a nuclear war. It is so great! It is so tremendous!

Despite the great and best success of Donald's Administration, there are still a few things he likes to complain about

> The Mexican government is much smarter, much sharper, and much more cunning. And they send all these bad hombres over because they don't want to pay for them. They don't want to take care of them.
>
> Donald J. Trump, First Republican Presidential Debate, August 6, 2015

He also frequently compares `Mexicans` to other bad people (like `Germans`, since they are exporting so many expensive cars to the US). Due to the tremendous amount of statements he has made (mostly containing less than 140 characters ...) the "Fake-News" New York Telegraph (NYT) has to put in a lot of effort to clarify and comment on all the statements of Donald. To check a statement,they have a list of facts they deem to be true and classify Donald's statements into three groups: real facts (which are logical conclusions from their list of true facts), exaggerations (which do not follow, but are still consistent with the papers list of facts), and alternative facts (which contradict the knowledge of the newspaper).

They have asked you to write a program helping them to classify all of Donald's statements – after all it is hard for a journalist to go through them all and check them all, right?

Input

The input consists of:

- one line containing two integers n and m, where
  - n ($1 \leq n \leq 200$) is the number of facts deemed true by the NYT;
  - m ($1 \leq m \leq 200$) is the number of statements uttered by the Donald.
- n lines each containing a statement deemed true by the NYT.
- m lines each containing a statement uttered by the Donald.

All statements are of the form a `are worse than` b, for some strings a and b, stating that a is (strictly) worse than b. The strings a and b are never identical. Both a and b are of length between 1 and 30 characters and contain only lowercase and uppercase letters of the English alphabet.

Note that Donald's statements may contain countries that the NYT does not know about. You may assume that worseness is transitive and that the first n lines do not contain any contradictory statement. Interestingly, Donald's press secretary (Grumpy Sean) has managed to convince him not to make up countries when tweeting, thus the input mentions at most 193 different countries.

Output

For every of the m statements of Donald output one line containing

- `Fact` if the statement is true given the n facts of the NYT.

- `Alternative Fact` if the inversion of the statement is true given the n facts of the NYT.
- `Pants on Fire` if the statement does not follow, but neither does its inverse.

Example

Input	Output
4 5 Mexicans are worse than Americans Russians are worse than Mexicans NorthKoreans are worse than Germans Canadians are worse than Americans Russians are worse than Americans Germans are worse than NorthKoreans NorthKoreans are worse than Mexicans NorthKoreans are worse than French Mexicans are worse than Canadians	Fact Alternative Fact Pants on Fire Pants on Fire Pants on Fire

Problem E. Perpetuum Mobile

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

The year is 1902. Albert Einstein is working in the patent office in Bern. Many patent proposals contain egregious errors; some even violate the law of conservation of energy. To make matters worse, the majority of proposals make use of non-standard physical units that are not part of the metric system (or not even documented). All proposals are of the following form:

- Every patent proposal contains n energy converters.
- Every converter has an unknown input energy unit associated with it.
- Some energy converters can be connected: If converter a can be connected to converter b such that one energy unit associated with a is turned into c input units for b, then this is indicated by an arc $a \xrightarrow{c} b$ in the proposal. The output of a can be used as input for b if and only if such an arc from a to b exists.

Einstein would like to dismiss all those proposals out of hand where the energy converters can be chained up in a cycle such that more energy is fed back to a converter than is given to it as input, thereby violating the law of conservation of energy

Einstein's assistants know that he is born for higher things than weeding out faulty patent proposals. Hence, they take care of the most difficult cases,while the proposals given to Einstein are of a rather restricted form: Every *admissible* patent proposal given to Einstein does not allow for a cycle where the total product of arc weights exceeds 0.9. By contrast, every *inadmissible* patent proposal given to Einstein contains a cycle where the the number of arcs constituting the cycle does not exceed the number of converters defined in the proposal, and the total product of arc weights is greater or equal to 1.1.

Could you help Einstein identify the inadmissible proposals?

Input

The input consists of:

- one line with two integers n and m, where
  - n $(2 \leq n \leq 800)$ is the number of energy converters;
  - m $(0 \leq m \leq 4000)$ is the number of arcs.
- m lines each containing three numbers a_i, b_i, and c_i, where
  - a_i and b_i $(1 \leq a_i, b_i \leq n)$ are integers identifying energy converters;
  - c_i $(0 < c_i \leq 5.0)$ is a decimal number

  indicating that the converter a_i can be connected to the converter b_i such that one input unit associated with a_i is converted to c_i units associated with bi. The number c_i may have up to 4 decimal places.

Output

Output a single line containing `inadmissible` if the proposal given to Einstein is inadmissible, `admissible` otherwise.

Example

Input	Output

2 2
1 2 0.5
2 1 2.3
	inadmissible

2 2
1 2 0.5
2 1 0.7
	admissible

Problem F. Plug It In!

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

Adam just moved into his new apartment and simply placed everything into it at random. This means in particular that he did not put any effort into placing his electronics in a way that each one can have its own electric socket.

Since the cables of his devices have limited reach, not every device can be plugged into every socket without moving it first. As he wants to use as many electronic devices as possible right away without moving stuff around, he now tries to figure out which device to plug into which socket. Luckily the previous owner left behind a plugbar which turns one electric socket into 3.

Can you help Adam figure out how many devices he can power in total?

Input

The input consists of:

- one line containing three integers m, n and k, where
  - m $(1 \leq m \leq 1500)$ is the number of sockets;
  - n $(1 \leq n \leq 1500)$ is the number of electronic devices;
  - k $(0 \leq k \leq 75000)$ is the number of possible connections from devices to sockets.
- k lines each containing two integers x_i and y_i indicating that socket x_i can be used to power device y_i.

Sockets as well as electronic devices are numbered starting from 1.

The plugbar has no cable, i.e. if it is plugged into a socket it simply triples it

Output

Output one line containing the total number of electrical devices Adam can power.

Example

Input	Output

3 6 8
1 1
1 2
1 3
2 3
2 4
3 4
3 5
3 6
	5

4 5 11
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
4 4
4 5
	5

3 5 7
1 1
1 2
2 2
2 3
2 4
3 4
3 5
	5

Problem G. Water Testing

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

You just bought a large piece of agricultural land, but you noticed that – according to regulations you have to test the ground water at specific points on your property once a year. Luckily the description of these points is rather simple. The whole country has been mapped using a Cartesian Coordinate System (where (0,0) is the location of the Greenwich Observatory). The corners of all land properties are located at integer coordinates according to this coordinate system. Test points for ground water have to be erected on every point inside a property whose coordinates are integers.

Input

The input consists of:

- one line with a single integer n $(3 \leq n \leq 100000)$, the number of corner points of your property;
- n lines each containing two integers x and y $(-10^6 \leq x, y \leq 10^6)$, the coordinates of each corner.

The corners are ordered as they appear on the border of your property and the polygon described by the points does not intersect itself.

Output

The number of points with integer coordinates that are strictly inside your property.

Example

Input	Output

4
0 0
0 10
10 10
10 0
	81

Problem H. Ratatöskr

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

Ratatöskr is a squirrel that lives in a giant (but finite) mythical tree called Yggdrasil. He likes to gossip, which sets the other inhabitants of the tree against each other. Ratatöskr is thus hunted by the two ravens of Odin, which are called Hugin and Munin, to bring him to justice.

The tree is made up of nodes connected by branches. Initially, the ravens and the squirrel sit on three different nodes. Now the following happens repeatedly:

- On Odin's signal, one of the ravens launches into the air and flies to another node of the tree (or possibly back to its previous position), while the other stays where it is.
- During this maneuver, Ratatöskr can travel along the branches to reach another node, but may not pass through a node where a raven sits. He is much quicker than the ravens and will reach his destination before the flying raven lands.

Ratatöskr gets captured if one of the ravens flies to his position and there is no other node he can escape to.

Help Odin determine an optimal strategy for capture, i.e. the minimum number of signals he has to give until Ratatöskr is guaranteed to be captured by a raven.

Input

The input consists of:

- one line with a single integer n ($3 \leq n \leq 80$), the number of nodes in the tree. The nodes are labeled $1, ..., n$.
- one line with a single integer r ($1 \leq r \leq n$), the initial position of the squirrel Ratatöskr.
- one line with a single integer h ($1 \leq h \leq n$), the initial position of the raven Hugin.
- one line with a single integer m ($1 \leq m \leq n$), the initial position of the raven Munin.
- $n-1$ lines each containing two integers i and j ($1 \leq i < j \leq n$), indicating a branch between nodes i and j.

The positions r, h and m are distinct. There is at most one branch between any two nodes and every node is reachable from every other node by a sequence of branches.

Output

One line containing an integer s, the number of signals that the ravens need to capture Ratatöskr in an optimal strategy. If Ratatöskr can escape them indefinitely, output `impossible`.

Example

Input	Output

4
1
2
4
1 4
2 4
3 4
	1

4
1
2
3
1 4
2 4
3 4
	2

Problem I. Überwatch

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

The lectures are over, the assignments complete and even those pesky teaching assistants have nothing left to criticize about your coding project. Time to play some video games! As always, your procrastinating self has perfect timing: Cold Weather Entertainment just released *Überwatch*, a competitive first person video game!

Sadly, you aren't very good at these kind of games. However, Überwatch offers more than just skill based gameplay. In Überwatch you can defeat all opponents in view with a single button press using your ultimate attack. The drawback of this attack is that it has to charge over time before it is ready to use. When it is fully charged you can use it at any time of your choosing. After its use it immediately begins to charge again.

With this knowledge you quickly decide on a strategy:

- Hide from your opponents and wait for your ultimate attack to charge.
- Wait for the right moment.
- Defeat all opponents in view with your ultimate attack.
- Repeat.

After the game your teammates congratulate you on your substantial contribution. But you wonder: How many opponents could you have defeated with optimal timing?

The game is observed over n time slices. The ultimate attack is initially not charged and requires m time slices to charge. This first possible use of the ultimate attack is there fore in the $(m+1)$-th time slice. If the ultimate attack is used in the i-th time slice, it immediately begins charging again and is ready to be fired in the $(i+m)$-th time slice.

Input

The input consists of:

- one line with two integers n and m, where
  - n $(1 \leq n \leq 300000)$ is the game duration;
  - m $(1 \leq m \leq 10)$ is the time needed to charge the ultimate attack in time slices.
- one line with n integers x_i $(0 \leq x_i \leq 32)$ describing the number of opponents in view during a time slice in order.

Output

Output the maximum number of opponents you can defeat.

Example

Input	Output

`4 2`
`1 1 1 1`
	`1`

`9 3`
`1 1 2 2 3 2 3 2 1`
	`5`

Problem J. Chocolate Gifts

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

This is Timothy's last semester here at UCF, and he is determined to finish college as the most favorite Knight ever! With his "extensive" friendship knowledge, he has determined that chocolate brings smile to any face and that is the key to becoming the most favorite Knight. So, Timothy has decided to send a chocolate bar to each and every UCF student. Specifically, Timothy will send a rectangular chocolate bar (made of smaller rectangular chocolate pieces assembled together) to as many UCF students as he can.

However, Timothy realizes that if two students see that they both received the same chocolate gifts, they would be repulsed and find this otherwise kind act suddenly creepy. To combat this, Timothy has decided that each gift he sends must be different. Thus, no pair of chocolate bars can have the same dimensions. But, because the smaller pieces have designs on them, a 5-by-3 chocolate bar is considered different than a 3-by-5 bar. Note, however, that a 5-by-5 (i.e., square) bar is the same as another 5-by-5 bar even though the smaller pieces have designs on them.

Timothy realized that he could of course send as many students unique chocolate bars as he wants because there are obviously an infinite number of sizes of chocolate bars he could make. However, because Timothy is no longer on the programming team, he is not that rich anymore! A chocolate bar with width a and height b will cost Timothy $a \cdot b$ and he has limited savings in his bank account. Thus, he can only afford a certain number of chocolate bars. More specifically, the total cost for the bars cannot exceed his savings. He thus needs your help to determine, given his limited budget, the maximum number of students he could send chocolate bars to such that no pair of students receive chocolate bars with the same dimensions.

In addition to all of this, Timothy must send the chocolate bars in nicely wrapped boxes, each bar in a separate box (i.e., not all bars in one box). But, all the boxes have the same size, i.e., there are not several boxes with varying sizes to choose from. Thus, every chocolate bar he makes must fit into one box. Furthermore, when putting a bar in a box, the bar cannot be rotated as this would cause the pattern on the chocolate to not match the pattern on the box. For example, let's assume the box has width 3 and height 5, then a 3-by-5 bar would fit in the box but a 5-by-3 bar will not fit in the box (again, the bar cannot be rotated).

Given Timothy's savings balance, determine the maximum number of rectangular chocolate bars he could make, such that no two bars have the same dimensions. Note that Timothy does not need to use up all of his money, but he cannot (of course) exceed his savings balance. Note also that each bar must fit into the given box size, i.e., the box size also restricts the bars he can make.

Input

The input consists of a single line containing three positive integers, w, h, and x ($1 \leq w, h, x \leq 10^{18}$) representing (respectively) the width and height of each box and Timothy's savings balance.

Output

Output a single line containing an integer representing the maximum number of students Timothy can send chocolate bars to given his constraints.

Example

Input	Output
2 1 2	1
3 2 11	4
8 3 64	13

Problem K. Cupcake Bonuses

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

The Unsweet Cupcake Factory (UCF) has recently seen a sudden increase in sales and business has been booming. Thus, the company has been rapidly expanding and is hiring new people nearly every day. Furthermore, to increase company morale (because the sugarless cupcakes don't seem to be helping), UCF has been paying out bonuses to employees belonging to certain departments according to each individual's performance.

The company is structured as follows:

- Each employee has exactly one direct supervisor (except for the CEO who has no supervisor).
- An employee can have zero or more direct subordinates (employees who have him/her as their supervisor).
- Each employee heads their own department.
- Every employee is a part of their own department and all of the departments that their supervisor is a part of (thus, the CEO is a part of only one department and all employees are a part of that department).

When UCF decides that a certain department is doing well, it pays out bonuses to all the employees that are a part of that department. Bonuses are calculated by multiplying the bonus amount B with the employee's bonus multiplier M. All employees begin with the same bonus multiplier, but depending on performance, an employee's bonus multiplier can change thus potentially changing the amount of money that employee gets for future bonuses.

UCF has found it to be quite a nuisance to handle the sudden increase in staff size. Thus, they have enlisted you to create a program which can help keep track of the amount of money paid out to employees in bonuses.

Given different queries, keep track of the amount of money paid to the different employees in bonuses. Queries will be one of four types:

1. Hire a new employee and assign him/her their supervisor.
2. Update an employee's bonus multiplier.
3. Pay out bonuses to all employees in the department headed by a given employee.
4. Retrieve (Display) the total amount of money paid to a given employee.

Employees are numbered starting at 1 (the CEO) and all new employees are given the next integer employee identification (id) number in the order they were hired into the company. Initially, the company has only the CEO, i.e., only one employee.

Input

The input will begin with a line containing two integers, n and S ($1 \leq n \leq 10^5$; $0 \leq S \leq 10^6$), representing (respectively) the number of queries to follow and the starting bonus multiplier for all new hires (including the CEO). The next n lines describe the queries and each is in one of the following four formats:

- "1 i" meaning that a new employee is hired and has the supervisor with employee id i.
- "2 i M" meaning that the bonus multiplier of the employee with the id i is now equal to M ($0 \leq M \leq 10^6$).
- "3 i B" meaning that a bonus with the base amount, B ($0 \leq B \leq 10^6$), is paid out to all employees in the department headed by i. Note that the bonus for an employee is calculated by multiplying the bonus amount B with the employee's bonus multiplier M.
- "4 i" representing a query asking for the amount of money paid in bonuses so far to the employee with the id i.

Assume that all the input values are valid, e.g., when referring to a supervisor (or an employee), assume that the supervisor (or the employee) exists.

Note that there will not be a query of Type-1 to indicate the hiring of CEO, i.e., assume the company starts with one employee (CEO).

Output

For each Type-4 query, print out a single line containing an integer representing the amount of money paid in bonuses to the employee in question thus far.

Example

Input	Output

7 1
3 1 10
4 1
2 1 2
1 1
3 1 5
4 1
4 2
	10
	20
	5

13 10
1 1
1 1
2 2 20
3 1 5
4 1
4 2
4 3
1 2
3 2 7
4 1
4 2
4 3
4 4
	50
	100
	50
	50
	240
	50
	70