Problem A. Christmas Tree Adapter

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

Adapters (Transformers) are used with many devices (e.g., cell phones) to convert the 110 volt coming out of the outlet. The adapter for Dr. Orooji's Christmas Tree broke and Dr. O could not find that model online or in stores. The tree adapter was showing the "ampere" but the adapters online were showing "watt" and "volt"! So, Dr. O had to refer to the "Electricity 101 Book" to figure out what to buy:

Watt: measure of power
Volt: measure of electric potential
Ampere: measure of current

watt = ampere × volt → ampere = watt / volt

Given the ampere for Dr. O's Christmas tree, and the watt and volt for a candidate adapter, determine if the candidate adapter will work with the tree. The adapter will work if its ampere is greater than or equal to the tree's ampere.

Input

The first input line contains an integer, a ($1 \leq a \leq 20$), indicating the ampere for the Christmas tree. The second input line contains two integers: w ($1 \leq w \leq 2000$), indicating the watt for the candidate adapter and v ($1 \leq v \leq 100$), indicating the volt for the candidate adapter. Assume that the input will not result in fractions in divisions.

Output

Print 1 if the candidate adapter will work with the tree, 0 (zero) otherwise.

Example

Input	Output
5 60 10	1
7 80 20	0
3 120 40	1

Problem B. Cool Phone Numbers

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

Phone numbers in USA are of the form `ddd-ddd-dddd`, i.e., 10 digits. In general, the fewer distinct digits in a phone number, the easier it is to remember the number. As a result, phone numbers with repeating digits are more popular.

Given a phone number, determine how many distinct digits there are in the number.

Input

There is only one input line; it provides the phone number using the above format. The input starts in column 1 and ends in column 12 (10 digits and 2 hyphens).

Output

Print 1-10 (the number of distinct digits) as follows:

1 - if all ten digits are the same, i.e., only one digit in the number, e.g., 888-888-8888
2 - if only two different digits in the number, e.g., 757-577-7577
3 - if only three different digits in the number
4 - if only four different digits in the number
5 - if . . .
⋮
10 - if ten different digits in the number, e.g., 246-189-0537

Example

Input	Output
888-888-8888	1
757-577-7577	2
246-189-0537	10
012-987-9690	7
000-000-0000	1

Problem C. Ready for Contest

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

Whenever the UCF Programming Team Coaches prepare for the UCF Local Programming Contest, each problem requires valid judge solutions in C++ and Java. Python solutions may be written as well, but these are optional. Thus, the coaches consider a problem to be ready for the contest if it has at least one C++ solution and at least one Java solution.

Unfortunately, instead of writing solutions for the problems that need them, the coaches just pick their favorite problems and write solutions for those instead. Naturally, they use their favorite language as well instead of checking to see what solutions hasn't been written yet.

Although many solutions have been written, it may be the case that some problems aren't ready for the contest yet. Write a program to determine which problems are ready to go!

Given the number of problems that have been proposed, and a list of every solution written (which problem and which language), determine the problems that are contest ready (have both a C++ and a Java solution written).

Input

The first input line contains two integers: n ($1 \le n \le 10^5$), indicating the number of problems proposed and m ($2 \le m \le 3 \cdot 10^5$), indicating the number of solutions written. The problems are numbered 1 through n.

The information about each solution written are provided in the following m input lines, with each line specifying a single solution written. Each of these input lines contains two integers: p ($1 \le p \le n$), and L ($1 \le L \le 3$), indicating that a solution has been written for problem number p in language L. If $L = 1$, the language is C++; if $L = 2$, the language is Java; if $L = 3$, the language is Python.

Output

Print a list of the problems that are ready for contest, sorted in numerical order, with one number outputted per line. It is guaranteed that at least one problem will be ready for contest so there will be some output.

(Example Input/Output on the next page)

Example

Input	Output

3 11
1 1
1 1
3 1
3 1
3 1
2 2
1 2
2 1
3 1
2 3
2 2
	1
	2

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

3 9
3 3
1 3
2 2
3 2
3 1
2 1
2 3
1 2
1 1
	1
	2
	3

Problem D. Fixing the Tournament

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

The most common type of tournament is a single elimination tournament that occurs in rounds. In order to run one of these tournaments, the total number of teams initially included must be a perfect power of 2. For example, if a tournament starts with 16 (2^4) teams, in the first round there will be 8 matches, resulting in 8 winning teams and 8 losing teams. In the second round, the 8 winning teams from the first round get paired up into four match ups. The process of pairing the winners continues until the final round when two teams remain and play in the championship match.

For the purposes of this problem, assume that:

- the teams are ranked from 1 to 2^n, for a positive integer n
- the best team is ranked 1 and the worst team is ranked 2^n (this is referred to as "one-based ranking")
- any time a team with a better rank (lower numbered ranking) plays a team with a worse ranking, the better ranked team wins.

The only thing not set for a tournament is which teams are going to play which teams in each round. You've hacked into the tournament software and can choose any set of pairings of the remaining teams for the matches in each round. Naturally, you'd like to figure out the latest round in the tournament your favorite team might make it to!

Given the number of rounds for a single elimination tournament and your team's one-based ranking, determine the highest round number your team could reach if you are able to create the match ups for each round optimally.

Input

There is only one input line; it contains two integers: n ($2 \leq n \leq 20$), indicating the number of rounds for the tournament and r ($1 \leq r \leq 2^n$), indicating your team's one-based ranking.

Output

Print, on a line by itself, the highest number round your team could reach.

Example

Input	Output
4 13	3
6 64	1
20 2	20

Problem E. Identical Letters

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

Streak of identical letters always fascinates computer scientists and, as such, the scientists always look for such consecutive sequence of identical letters.

Given a string of lowercase letters and an integer m, determine the maximum number of consecutive identical letters in the string if you can remove up to m letters from the string. Note that you do not have to remove exactly m letters.

Input

The first input line provides the string ($1 \leq$ string length $\leq 2 \cdot 10^5$); it starts in column 1 and contains only lowercase letters. The second input line contains an integer, m ($0 \leq m \leq$ string length), indicating the maximum number of letters you can remove from the string.

Output

Print the maximum number of consecutive identical letters in the string if you can remove up to m letters from the string.

Example

Input	Output

`bbazbcbbbcybbx`
`2`
	`5`

`bbazbcbbbcybbx`
`5`
	`8`

`zabcadyhxwuy`
`5`
	`2`

Explanation

For the first Example Input, we can remove the two letters at positions 10 and 11.

For the second Example Input, we can remove the letters at positions 3, 4, 6, 10 and 11.

For the third Example Input, we can create "`...aa...`" or "`...yy...`", each of length 2.

Problem F. Espresso Made Your Way

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

You have a new job. You can honestly say that coffee is not your cup of tea. You have many customers that ask for a particular ratio of coffee to milk. You don't have the ability to make arbitrary measurements. You have two cups. Both cups are the same size. You have a machine that dispenses pure coffee, and another machine that dispenses pure milk.

You can completely fill one empty cup with coffee and the second empty cup with milk. You can also take these two full cups and create a mix such that the result will have exactly half of each cup, i.e., half coffee and half milk. So, you have created a mix (a cup) that is half coffee and half milk but, unfortunately, you have wasted a full cup of ingredients (the second half of coffee and the second half of milk).

Now that you have a cup that is half coffee and half milk, you can fill the second cup with coffee or fill the second cup with milk. Then, you can mix your two cups to create a cup with a different coffee-to-milk ratio. Note that, when mixing two cups, you use exactly half of each cup and waste exactly half of each cup.

By repeating the above process, you can make some coffee-to-milk ratios but some ratios are literally impossible.

Given the tolerance range a customer has for their coffee-to-milk ratio, determine the least amount of ingredients you will waste assuming you prepare the customer's order optimally.

Input

There is only one input line; it contains four integers: C_1, M_1, C_2, and M_2 ($0 \leq C_1, M_1, C_2, M_2 \leq 10^6$) representing the coffee-to-milk ratio range the customer accepts. More specifically, the ratio can be anywhere from C_1 coffee parts per M_1 milk parts to C_2 coffee parts per M_2 milk parts. You are guaranteed that:

- The two ratios will not be identical.
- $C_1 + M_1 > 0$
- $C_2 + M_2 > 0$
- $C_1/(C_1 + M_1) \neq C_2/(C_2 + M_2)$

Output

Print one integer representing the least number of cups of ingredients wasted assuming you prepare the customer's order optimally.

Example

Input	Output
`1 2 2 1`	`1`
`1 10 0 1`	`0`
`1 99 49 51`	`2`

Problem G. Speed Ups

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

In a video game called Speed Ups, you are a single player running a race. The goal is to finish the race as fast as possible. Your regular speed is 1 meter/second. But at different meter markers, you may get the choice of a speed up. Each speed up can be described by an ordered triplet of integers $< x,\ m,\ d >$ indicating that the speed up can be used "exactly" x meters after the start of the race, giving you a temporary speed of m meters per second for a duration of d seconds. If you are currently using a speed up, you cannot use another speed up. Thus, it's possible that if you choose to use one speed up, you'll run past the location of another speed up which might have been better to take.

For example, imagine that for a 100 meter race, there are two possible speed ups: $< 10,\ 2,\ 5 >$ and $< 15,\ 3,\ 20 >$. You take 10 seconds to run the first 10 meters. Then, if you take the first speed up at the 10 meter mark, you will then run the next 5 seconds at 2 meters/second, arriving at the 20 $(10 + 5 \cdot 2)$ meter mark 15 $(10 + 5)$ seconds after the start of the race. Since this (20 meter mark) is further along than the second speed up (15 meter mark), you can no longer take the second speed up and will complete the race in $10 + 5 + 80 = 95$ seconds. If however, you choose to skip the first speed up at the 10 meter mark, but take the second one at the 15 meter mark, you'll then travel at a speed of 3 meters/second for 20 seconds, arriving at the 75 meter mark 35 seconds into the race, finishing the race in 60 seconds $(15 + 20 + 25 = 60)$, which is the fastest time in which you could complete the race for this example.

Given the length of the race and a list of all the potential speed ups (location, new speed, duration), determine the fastest possible time (in seconds) that you could finish the race.

Input

The first input line contains two integers: n $(1 \leq n \leq 1000)$, indicating the number of possible speed ups available and L $(1 \leq L \leq 10^9)$, indicating the length of the race in meters.

Information about the speed ups is provided in the following n input lines, one speed up per line. Each of these input lines contains three integers: x $(1 \leq x \leq L - 1)$, m $(2 \leq m \leq 100)$ and d $(1 \leq d \leq 10^6)$, denoting that there is a speed up located x meters from the start of the race that will change your speed to m meters a second for a duration of d seconds. Note that:

- There could be multiple speed ups located at the same exact location but you can take only one of them.
- If you take a speed up, you must use it until its completion, or the end of the race, whichever comes first.
- Once you choose to use a speed up, you can't use another one until the previous one is fully completed.
- If one speed up puts you at a distance y from the start and there's a new speed up at exactly y from the start, you can take that new speed up.

Output

Print, on a line by itself, the fastest time in seconds you could possibly finish the race. Any answer within an absolute or relative tolerance of 10^{-6} will be accepted.

Example

Input	Output
2 100	60.0
10 2 5	
15 3 20	
3 1000	550.0
25 3 25	
100 2 400	
25 5 20	
1 50	17.75
7 4 200	

Explanation

The first example is explained in the problem description above.

In the second example, it's better to take the first speed up instead of the third one listed because taking the third speed up prevents you from taking the second speed up. Thus, after 25 seconds, if you take the first speed up, you'll travel at 3 meters per second for 25 seconds, arriving at the 100 meter mark 50 seconds into the race. Then, you can take the second speed up (had it been located at $x = 99$ you would not have been able to take it) and travel at 2 meters/second for 400 seconds, ending up at the 900 meter mark 450 seconds into the race. You have to then finish the race running 1 meter/second for the last 100 seconds.

In the last example, when you take the first speed up after 7 seconds, you travel at 4 meters per second for the rest of the race (43 meters) which will take 10.75 more seconds. This example illustrates the situation where the race ends before the entire duration of the speed up taken.

Problem H. Magnetic Attractions

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

You have a new job. You are working with magnets. You have a metal bearing that will lie on a 2D plane. You have two magnets: one weak and one strong. You know that the force exerted upon an object by a magnet is inversely proportional to the square of the distance between the two objects, but linearly proportional to the strength of the magnet. The formula is:

$$force = strength/distance^2$$

Given the locations of two magnets in the 2D plane and the strength of each magnet, determine the area of the region in which a metal ball bearing can be placed in the grid, such that the force of the weaker magnet upon the bearing is more than the force of the stronger magnet, i.e., the bearing is more attracted to the weaker magnet than the stronger magnet.

Input

The first input line contains three integers: s, x, y ($1 \leq s, x, y \leq 300$), representing the strength, x and y coordinate of the weaker magnet. The second input line contains three integers: S, X, Y ($1 \leq S, X, Y \leq 300$), providing the information for the stronger magnet following the same format as the first input line. You are guaranteed that $s < S$ and that the two magnets will not be in the same location.

Output

Print a single number representing the area of the region in which a bearing could be placed such that the force between the bearing and the weak magnet is more than the force between the bearing and the strong magnet. Any answer within an absolute or relative error of 10^{-6} will be accepted.

Assume that the region is convex and bounded. Note that the region should not include the area where the forces between the bearing and the two magnets are equal.

Example

Input	Output
3 10 7 6 9 8	12.566371
5 5 5 6 6 6	188.49556

Problem I. Hotel Rooms

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

When the UCF Programming Team travels, the coaches would like to get hotel rooms that are close to each other. There is a hotel where rooms are numbered 1 through n and these rooms are in a straight line, i.e., Room 2 is next to Room 1, Room 3 is next to Room 2, and so on. So, it is easier to find large number of available rooms that are close to each other.

Given the room reservations, you are to determine the availability of rooms to accommodate the UCF Programing Team (a large group).

Input

The first input line contains two integers: n ($1 \leq n \leq 5 \cdot 10^5$), indicating the number of hotel rooms and t ($1 \leq t \leq 10^5$), indicating the number of transactions. Each of the next t input lines contains a transaction to be processed. There will be two types of transactions:

- Room Reservation: This input line starts with the letter R in the first column, followed by one space, followed by a valid room number. This transaction is reserving the given room (assume that the room is not already reserved).
- Group Room Availability: This input line starts with the letter A in the first column, followed by one space, followed by a valid starting room number, followed by a space, followed by a valid ending room number. This transaction is asking how many rooms are available in the given range. Assume that the ending room number will not be less than the starting room number, i.e., the requested range is valid.

Output

There is no output required for the room-reservation transactions. For each group-room-availability transaction, output a separate line providing the total number of available rooms in the requested range.

Example

Input	Output
20 8 A 5 10 R 6 R 9 R 3 A 5 10 R 13 R 18 A 1 20	6 4 15

Problem J. Password Protection

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

You have recently made an account on a website. You used a password manager to produce a random password. One of the constraints was that your password does not contain your first or last name. You assumed that containment here referred to having a substring (contiguous sequence of characters) that were identical to the sequence of characters that comprised either your first or last name.

You were unlucky enough that your first password did not satisfy this criterion. You want to know how often such an unfortunate turn of events could happen. Write a program to help in this endeavor by simply counting the number of passwords that contain either a given first name or last name as a substring. For example, if your first name was "ali" (quotes for clarity), then the password "california" contains your first name as a substring but the password "applied" does not contain your first name as a substring, even though "ali" is a subsequence of this password.

Note that this is an "inclusive or" which means either or both, i.e., a password is not valid if it contains the first name or the last name or both.

Given a first name, last name, and password length, determine the number of randomly generated passwords that will contain your first or last name as a substring (contiguous subsequence of characters). You can assume that passwords generated will contain only lowercase letters.

Input

The first input line contains an integer, n ($1 \leq n \leq 10^7$), representing the length of the password that is randomly generated. The second input line contains a string of x lowercase letters ($1 \leq x \leq 128$), representing the first name to check. The third input line contains a string of y lowercase letters ($1 \leq y \leq 128$), representing the last name to check.

Output

Print an integer representing the number of invalid passwords, i.e., they contain the first or last name. Since the answer can be quite large, your program should report the number as the smallest non-negative remainder when divided by $10^9 + 7$.

Example

Input	Output
6 ali orooji	70304
5 arup guha	104

Problem K. Team Work

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

The success of the UCF Programming Teams over 40+ years are attributed to the talented and dedicated students and to the experienced and great coaches. The coaches teach the various algorithms and team strategies, and the students deliver in various competitions.

A team in the International Collegiate Programming Contest consists of three students. One of the major team strategies is "who should do what problem". Wouldn't it be nice if the coaches had a program to answer this? And, wouldn't it be even nicer if you wrote the program for the coaches?!

Given the following information:

- number of students on a team,
- number of problems in the set,
- who can solve what problems and how long it takes to solve each problem, and
- total time available (how long the contest is),

determine the maximum number of problems that can be solved by the entire group in the given time. To simplify the problem, assume that:

- A student is limited to solving at most one problem in the entire contest. For example, let's assume the contest is 5 hours and Student$_1$ can solve Problem$_7$ in 1 hour and Problem$_9$ in 2 hours. Even though this student can finish both problems in 3 hours (within the 5-hour contest), the student is still limited to at most one problem in the entire contest.
- There is only one computer for the entire team. So, if a student is using the computer, the teammates cannot use the computer, i.e., problems must be solved one at a time. For example, let's assume the contest is 5 hours, Student$_2$ can solve Problem$_6$ in 2 hours and Student$_3$ can solve Problem$_8$ in 4 hours. Since problems are solved on the computer one at a time, the 5-hour contest is not enough to finish both problems since the two problems require 6 hours. Again, the problems are not solved in parallel even though there are multiple students; the problems are solved one at a time.

Input

The first input line contains three integers as follows:

- n ($1 \leq n \leq 200$), number of students on the team,
- p ($1 \leq p \leq 200$), number of problems in the set,
- t ($1 \leq t \leq 10^5$), total time available (how long the contest is).

The next n input lines provide who can solve what problems and how long it takes for each problem. Each of these n input lines contains p integers; row$_1$ is the information for student$_1$, row$_2$ is for student$_2$, and so on. The first integer on a row shows how long it takes that student to solve problem$_1$, the second integer on a row shows how long it takes that student to solve problem$_2$, and so on. Each of these integers are between 0 and 10^5, inclusive. A value of zero indicates that student can't solve that problem (doesn't know how to solve that problem).

Output

Print the maximum number of problems that can be solved by the entire group in the given time.

Example

Input	Output

2 4 5
2 5 3 4
0 4 5 0
	1

2 4 5
1 5 1 1
2 1 0 1
	2

Problem L. Please Please Please

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

You aim to please, or at least you did. Now you are a teacher and you have students asking you to perform certain tasks some of which contradict each other. You want to be objective, but you also want to make your job easier. You will automate parts of your job. The first task is assigning letter grades at the end of the semester.

Several students have sent you emails stating their overall grade in the class as an integer and their desired letter grade. Normally, you would disregard such pleas but, in this case, they have used the magic word "please". Now you feel that you should cater to some of these requests.

For simplicity, we assume there are five letter grades: A, B, C, D, and F. You will assign letter grade cutoffs such that each letter grade is achievable by some integer overall grade in the range of 0 to 100, inclusive, even if there is no student with that particular integer score. Additionally:

- Any possible overall grade that earns an A should be higher than any possible overall grade that earns a B.
- Any possible overall grade that earns a B should be higher than any possible overall grade that earns a C.
- Any possible overall grade that earns a C should be higher than any possible overall grade that earns a D.
- Finally, any possible overall grade that earns a D should be higher than any possible overall grade that earns an F.

If a student asks for a letter grade, they will be disappointed if they receive any other letter grade (even a higher letter grade), e.g., if a student asks for a B, they will be disappointed if they receive a C or A or ... (any grade other than B).

Your grade assignment preference is as follows:

- You want to prioritize the student(s) that used the word please the most. That is, your letter grades will be assigned such that you maximize first the number of satisfied students that used please the most, then the students that used please the next most, and so on. Note that this criterion could result in satisfying only one student and making everyone else unhappy. Again, satisfying the student(s) that used please the most is the highest priority. Then, satisfying the students that used please the next most, and so on.
- If there are multiple assignments that can do the above, you want to choose the one that next maximizes the cutoff line for the A, i.e., the highest integer value for cutoff line for A.
- If multiple assignments for letter grades are still possible, you want to choose the one that maximizes the score for the B cutoff.
- If multiple cutoffs still exist, the one that maximizes the C cutoff should be selected next.
- Lastly, if multiple cutoffs still exist, the assignment that maximizes the D cutoff should be prioritized.

Given the people that have sent emails requesting particular letter grades, their corresponding score, and the number of times they used the word please, determine the best fit letter grade cutoffs following the criteria listed above.

Input

The first input line contains an integer, n ($1 \leq n \leq 10^5$), representing the number of students. The next n input lines provide information about the students, one student per line. Each of these input lines starts in column 1 and contains three tokens (separated by exactly one space) as follows:

- an integer, g ($0 \leq g \leq 100$), representing their grade,
- An uppercase letter, L (L is one of 'A', 'B', 'C', 'D', or 'F'), representing their desired letter grade,
- and an integer, p ($1 \leq p \leq 10^5$), representing the number of times they used the word "please" in their email.

Output

Print four integers on a single line representing the letter grade cutoffs for A, B, C, and D, respectively.

Example

Input	Output
5 90 A 1 88 C 5 85 B 3 50 D 2 89 F 3	90 89 88 50
3 90 B 5 90 A 4 90 A 4	100 90 89 88