Problem A. Corona Virus Testing

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

Testing for Corona can be done individually, e.g., 100 people require 100 test kits. Alternatively, the test can be done in groups (pools), e.g., 100 people can be divided into five group of 20 people each and then using only one test kit per group. If one or more groups test positive, then individual tests are needed for each person in those group. So, for our example, five groups will need 5 test kits and let's say two groups test positive, so we would need additional 40 (2*20) test kits for a total of 45 (5+40) test kits.

The Problem:

Given the data for the two possible testing approaches, determine which approach will use fewer test kits.

Input

There is only one input line; it provides three integers: g ($2 \leq g \leq 50$), indicating the number of groups, p ($2 \leq p \leq 50$), indicating the number of people in each group, and t ($0 \leq t \leq g$), indicating how many groups tested positive (i.e., people in these groups need to be tested individually).

Output

Print 1 (one) if testing everyone individually will use fewer kits, 2 (two) if testing in groups will use fewer kits, and 0 (zero) if the two approaches use the same number of kits.

Example

Input	Output
40 3 38	1
10 20 2	2
20 10 18	0

Problem B. Presidential Election

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

Presidential election is coming up (November). In 2016, Clinton won the "*majority*" votes but Trump ended up with more "*electoral*" votes and won the race. (As a reminder, if a candidate receives more votes in a state, that candidate wins all the electoral votes for that state, i.e., electoral votes for a state are not divided proportionally based on the votes received by each candidate in that state.)

The Problem:

Election is in less than two months so let's predict the outcome! Given the voting data for each state, determine who wins the majority votes and who wins the electoral votes.

Input

The first input line contains an integer, n ($1 \leq n \leq 50$), indicating the number of states. Each of the next n input lines contains three integers, providing voting data for a state: e ($1 \leq e \leq 100$), indicating electoral votes for the state, v_1 ($0 \leq v_1 \leq 1000$), indicating votes for the first candidate, and v_2 ($0 \leq v_2 \leq 1000$; $v_2 \neq v_1$), indicating votes for the second candidate.

Output

Print 1 (one) if the first candidate wins both the majority votes and the electoral votes. Print 2 (two) if the second candidate wins both the majority votes and the electoral votes. Print 0 (zero) for all the other cases. Assume that if the total majority votes for the two candidates tie, neither one wins the majority. Similarly, if the total electoral votes for the two candidates tie, neither one wins the electoral.

Example

Input	Output
3 5 10 50 15 30 60 10 25 15	2
3 5 48 50 15 57 60 10 25 15	0

Problem C. Microwave Mishap

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

Donald is very hungry. He grabs a TV dinner, puts it in the microwave, and enters 02:15 to cook it for 2 minutes and 15 seconds. Unknown to Donald, microwave takes these values as hours and minutes, i.e., microwave takes 02:15 as 2 hours and 15 minutes (not 2 minutes and 15 seconds). We need to tell Donald how much extra (i.e., the additional time) his food will be cooking. That is, we need to tell Donald that his food will cook 2 hours, 12 minutes, and 45 seconds more (longer) than what he was expecting!

The Problem:

Given the initial time in the form of *MM:SS*, where the input is actually taken as *HH:MM*, determine how much extra the food will be in the microwave. Provide this info in the form of *HH:MM:SS*.

Input

Input will consist of a single line in the form of *MM:SS*, representing what Donald has entered. *MM* and *SS* will be in the range of 00 and 59 (inclusive), but they both will not be 00 at the same time, i.e., the total time will be positive, i.e., input will not be 00:00.

Output

Output should contain a single line in the form of *HH:MM:SS*, indicating the additional time the food will cook in the microwave. Note that you need to print two digits for each part. Also note that *MM* and *SS* must be in the range of 00 to 59.

Example

Input	Output
05:00	04:55:00
13:37	13:23:23
00:10	00:09:50

Problem D. Food Display Arrangement

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

Your friend, Thomas, is working on Food Display Arrangements (FDA). He has all the food lined up on a long row (table). His job requires that he arranges the FDA in an aesthetically pleasing manner. An FDA is aesthetically pleasing if all the food of the same type is grouped together, i.e., all the food of the same type are next to each other. Thomas can reorganize the FDA as follows: pick up all the food of one type and place it on either end of the table, i.e., place it at the beginning of the table or at the end of the table. Thomas wants to know the minimum number of reorganization steps needed to make the FDA aesthetically pleasing. Note that you don't need to tell him the specific steps, only the least number of steps.

The Problem:

Given a display of food by their types, determine the minimum number of times necessary to move all food of the same type to the end or the beginning of the display to ensure that all food of the same type is grouped together. Assume that the display can be extended at the ends to contain any amount of moved food.

Input

The first input line contains an integer, n ($1 \leq n \leq 10^5$), representing the number of food items in the display. The next input line contains n space separated integers, a_i ($1 \leq a_i \leq 10^9$), representing the id of the i^{th} food item in the display.

Output

Print the minimum number of times necessary to move all food of the same type to the beginning or the end to make the FDA aesthetically pleasing.

Example

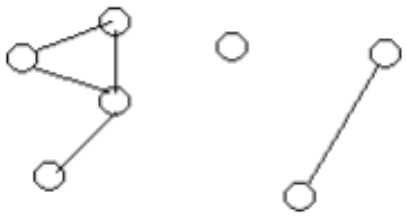
Input	Output
15 8 8 2 8 7 8 1 1 3 7 3 4 2 4 7	2
10 1 2 3 4 5 1 2 3 4 5	4

Problem E. Making Connections

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

Given that everything is online these days, connectivity is a must. A computer network can be modeled as a graph, where each computer is a vertex and each direct network connection between pairs of computers is an undirected edge.

Consider the process of building a computer network. At the very beginning there will be n computers, with no connections between any of them. Then, as time goes on, pairs of computers are chosen, one pair at a time, and a direct network connection between them is added. In the middle of such a process, we might get the following graph modeling the current connections:



This network currently has three groups: the first group has 4 computers that can communicate with each other directly or indirectly, the second group consists of 1 computer by itself, and the third group has 2 computers that can communicate with each other. So, a group is each of the separate components of the graph.

We can define the average connectivity of a network as the sum of the group sizes squared, divided by the number of components. For the example graph shown above, the current connectivity equals $(4^2 + 1^2 + 2^2)/3 = 21/3 = 7$.

As a network is being built, the project manager would like to know the average connectivity of the network at that given snapshot of time. Write a program to handle the queries as the network is being constructed.

The Problem:

Given a network of n initially separate computers, along with a sequence of steps, where each step is either a pair of computers being connected or a query about the average connectivity as of that moment, process each step (i.e., connect the two computers or answer the query).

Input

The first input line contains two space separated integers: n ($1 \leq n \leq 10^5$), representing the number of computers, and m ($1 \leq m \leq 3 \cdot 10^5$), representing the total number of steps (each step being either connect two computers or a query on the average connectivity as of that moment). Assume that the computers are numbered 1 through n, inclusive.

Each of the following m input lines contains information about one operation, in the order that they occur. Each of these lines will start with a single integer, either 1 or 2, providing the step type. If the step type is 1, it means that a pair of computers is being connected with a direct connection. The value of 1 will be followed by u and v ($1 \leq u,\ v \leq n,\ \ u \neq v$), representing the pair of computers being connected with a direct connection. If the step type is 2, this is a query and no other information will be on that input line.

Note: It's possible that the input contains multiple direct connections for the same pair of computers; the extra direct connections between the same two computers do not have any effects. It's also possible that, at the end of the process, not all n computers are connected in the same component, i.e., there may be more than one component even at the end of the process.

Output

For each query, output the average connectivity of the network at that point in time as a fraction in lowest terms on a line by itself. Specifically, output two integers, x and y, with the character '/' in between, indicating that the average connectivity of the network at the time is x divided by y such that x and y share no common factors, e.g., 21/12 is not correct (should be 7/4).

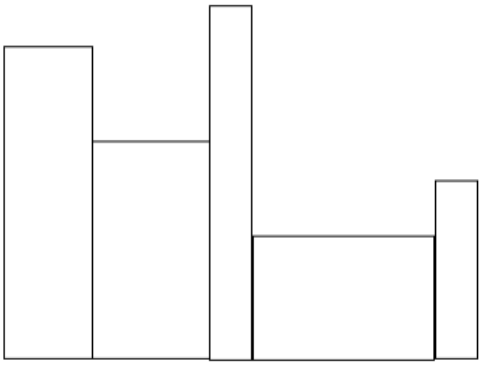
Example

Input	Output
7 9 2 1 1 2 1 1 3 2 1 3 4 1 2 3 2 1 6 7 2	1/1 13/5 19/4 7/1
4 9 1 1 2 2 1 3 4 2 1 2 3 2 1 1 4 1 2 4 2	2/1 4/1 16/1 16/1

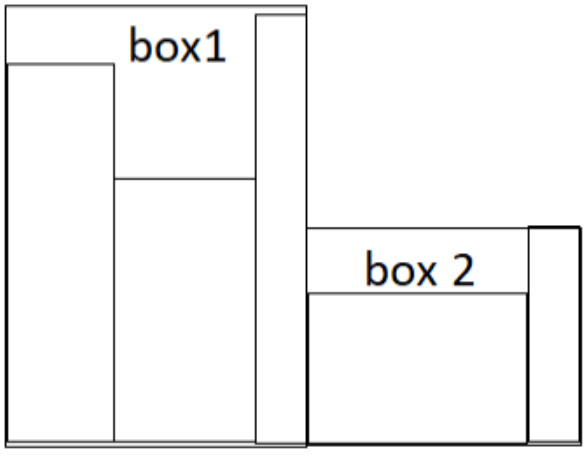
Problem F. Boxing Books

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

You have many books all in a row on a single long shelf and will need to box them up for a move. Each book is the same depth, but the books have different heights and widths. Here is a quick illustration of several books on a shelf:



Unfortunately, the moving company is limiting the number of boxes you can use and charges based on the height and width of each box. In addition, they are quite lazy and refuse to grab books from different areas of the shelf for each box, i.e., they only grab a sequence of adjacent books for each box. And, they don't want to change the orientation of any book (to save space) when placing it in a box. (Unfortunately, during the COVID-19 crisis, this was the only company you could find, so you'll have to deal with their idiosyncrasies for this problem.) Thus, all you are allowed to tell the moving company is how many adjacent books belong in each box, starting from the left part of the shelf. For example, if you were only allowed two boxes for the picture above, you could choose to put the first three books in the first box and the last two books in the second box as illustrated below:



The charge for each box is simply the height of the box times its width. As can be seen in the diagram, the height of each box is simply the maximum height of the books in that box and the width of the box is the sum of the widths of the books in the box.

Naturally, you would like to minimize the cost of boxing up the books given the ridiculous restrictions of the moving company. Write a program to do this.

The Problem:

Given a list of the dimensions of n books on a shelf, as well as the number of boxes, k, to put the books in, determine the minimum cost to box the books in exactly k non-empty boxes. Note that you must use exactly k boxes, not less and not more.

Input

The first input line contains two integers: n ($1 \leq n \leq 1000$), representing the number of books, and k ($1 \leq k \leq n$), representing the number of boxes to put the books in. The following n input lines contain the dimensions of the books, one book per line, in the order they appear on the shelf, from left to right. The i^{th} of these input lines contains two integers, w_i ($1 \leq w_i \leq 10^6$) and h_i ($1 \leq h_i \leq 10^6$), representing (respectively) the width and height of the i^{th} book on the shelf from the left.

Output

On a line by itself, print the minimum cost of boxing the books.

Example

Input	Output
5 2 3 10 4 7 1 12 6 4 1 6	138
5 5 2 6 1 8 3 4 2 12 3 9	83

Explanation of the first Example Input/Output: Putting the first three books in one box and the last two books in the second box will result in the minimum cost.

Box1: (3 + 4 + 1) * 12 = 96

Box2: (6 + 1) * 6 = 42

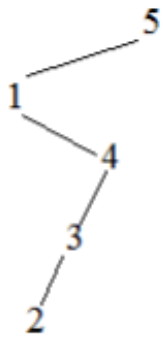
Total: 96 + 42 = 138

Problem G. Bad Tree

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

Binary Search Trees are supposed to speed up searches for items but this happens only when the height of the tree is much less than the total number of values stored in the tree. And, in programming contests, judges unfortunately try to make the "worst case data". So invariably, in binary search tree problems, judges will make data where n nodes get inserted into a tree in a particular order so that the height of the tree is the worst case, $n - 1$.

Without loss of generality, let's assume that the n items to be inserted into a binary search tree are 1, 2, 3, ..., n. For $n = 5$, if we insert the values in this order: 5, 1, 4, 3, and 2, we get the following binary search tree of height 4:



In fact, there are quite a few orderings of the first n positive integers that, when inserted into a binary search tree, create a binary search tree of height $n-1$. Since you aspire to be a great judge one day for this contest, write a program to generate the k^{th} lexicographical permutation of the first n positive integers that, when inserted into a binary search tree in that order, generates a binary search tree of height $n - 1$.

The Problem:

Given a positive integer n, and another positive integer k, determine the k^{th} permutation, in lexicographical ordering, that when the items are inserted into a binary search tree in that order, generates a tree of height $n - 1$.

Input

There is only one input line; it contains two space separated integers: n ($1 \leq n \leq 100$), representing the number of nodes in the binary search tree, and k ($1 \leq k \leq 10^{18}$), where we desire the k^{th} lexicographical permutation of the first n integers which creates a binary search tree of height $n - 1$, when inserted in the order given in the permutation.

Output

Print the k^{th} lexicographical permutation of the integers 1 through n of the permutations which, when the values are inserted into a binary search tree, create a tree of height $n - 1$. Output the permutation on a single line, following each number in the permutation with a space. If no such permutation exists, output -1 instead.

Example

Input	Output
5 12	5 1 4 3 2
4 2	1 2 4 3
6 1	1 2 3 4 5 6
3 50	-1

Problem H. Hang Gliding

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

The 2020 hang gliding world championships are coming to Florida next spring! (You may think it is odd to hold in Florida a sport that requires mountains, but it turns out that hang gliders can be towed to altitude by other aircraft.) The competition is divided into a set of tasks; completing a task successfully gives a pilot a number of points. Either all points are awarded for a task, or none are. For each task, every pilot has a probability of success. A pilot's score is the total of all successfully completed tasks at the end of the competition.

This year, the organizing committee couldn't decide on a reasonable number of tasks, so the time slots for tasks overlap. At any given time, there can be multiple tasks going on, but a pilot may only choose one to be competing in at that time. Each pilot may compete in as many tasks as they want given this constraint. The pilots know their own strengths and weaknesses, and will choose tasks in order to maximize their expected score.

You have been offered free hang gliding lessons if you help with scoring software for the event. Among other things, the committee wants the software to be able to predict the winners ahead of time.

Given a set of tasks, each with a time slot and a point score to be awarded for success, and a list of pilots, each with success probabilities for each task, compute the expected score for each pilot, and report the top 3 expected scores.

Input

The first input line contains two integers: t ($1 \leq t \leq 10^4$), indicating the number of tasks, and p ($3 \leq p \leq 100$), indicating the number of pilots.

The next t input lines contain the task descriptions. Each line contains three integers (s, e, and a) separated by a space: $0 \leq s < e \leq 10^4$, s is the start time of the task, and e is the end time of the task, in minutes after the competition starts; a ($1 \leq a \leq 100$) is the number of points awarded for the task. Note that the task start times and end times are non-inclusive, i.e., if a task ends at time T and another task starts at time T, a pilot can compete in both tasks.

After the task descriptions, there are t lines for each pilot. The first t lines in this section are the probabilities of success for each task for pilot 1; the next t lines are the probabilities of success for pilot 2, and so on. The probabilities are floating point numbers in the range 0 to 1, inclusive.

Output

Print the top 3 pilots. Print, in order of descending expected score, the pilot's number, a space, and the pilot's expected score, rounded to 2 decimal places (i.e., a score of 42.494 would be printed as 42.49; a score of 42.495 would be printed as 42.50). Note that pilots are numbered starting at 1, not zero.

Example

Input	Output
3 3 0 1 5 1 2 10 2 3 15 0.0 0.0 0.2 1.0 0.0 0.0 0.0 0.75 0.0	3 7.50 2 5.00 1 3.00
3 4 0 3 50 3 6 50 0 6 75 0.2 0.3 0.6 0.6 0.6 0.5 0.6 0.5 0.4 0.9 0.9 0.9	4 90.00 2 60.00 3 55.00

Problem I. Paragliders and Aircraft

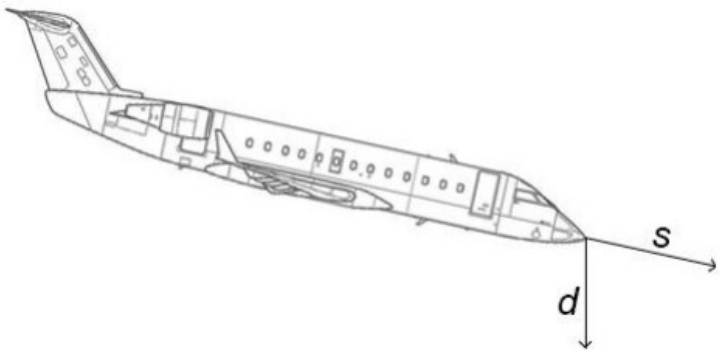
Source file name:	incoming.c, incoming.cpp, incoming.java, incoming.py
Input:	`Standard`
Output:	`Standard`

A friend of yours is a paraglider pilot who flies at Tiger Mountain outside of Seattle, WA. The site is popular and, unfortunately, also in the landing approach patterns of some flights heading to Sea-Tac airport or Boeing field. This could obviously lead to dangerous situations involving highflying paragliders and low-flying jets. Your friend, knowing you're a code guru, has asked for your help in developing an early warning system to let paraglider pilots know when jet traffic might intrude upon their airspace.

You've researched and found a web API that can give you real-time flight data on nearby air traffic, and another that lets you send text messages to the pilots. Now all you have to do is to figure out which flights could cause a problem.

The Problem:

Treat the airspace as a 3D coordinate space measured in feet. (This is reasonable because, for small distances, you can interpret latitude and longitude as coordinates on a plane.) You can assume the paragliders will stay in a 3D cylinder with a center, a radius, and lower and upper altitude bounds. Flight data will contain each aircraft's position, altitude, heading, velocity, and descent rate. Heading is in degrees where 0 degrees is along the positive x axis and 90 degrees is along the positive y axis. Airspeed is in feet per second, along vector s in the diagram; descent is along vector d.



If an aircraft's flight path will intersect the bounded cylinder, compute the entry and exit time in seconds from the current time, and output a message with those times. Otherwise, output a message to ignore the aircraft.

Input

The first input line describes the airspace; it contains five floating point numbers: the $(x_c,\ y_c)$ coordinates of the center of the cylinder in feet $(-5 \cdot 10^4 \leq x_c,\ y_c \leq 5 \cdot 10^4)$, the radius (r) of the cylinder in feet $(1 \leq r \leq 10^4)$, and the lower (l) and upper (u) bounds of the cylinder in feet $(0 \leq l < u \leq 10^4)$.

The second input line contains a single integer, n $(1 \leq n \leq 100)$, indicating the number of aircraft to process.

The next n input lines describe the incoming aircraft. Each line provides data about an aircraft and contains seven numbers: an integer flight number f $(1000 \leq f \leq 9999)$, the $(x_a,\ y_a)$ position of the aircraft $(-2 \cdot 10^5 \leq x_a, y_a \leq 2 \cdot 10^5)$, the heading (h) of the aircraft in degrees $(0 \leq h < 360)$, the altitude (a) in feet $(1000 \leq a \leq 35000)$, the speed (s) in feet per second $(100 \leq s \leq 10^4)$, and the descent rate (d) in feet per second $(0 \leq d \leq s)$. All inputs for the incoming aircraft, other than the flight number, are given in floating point.

Assume that the aircraft will never start inside the cylinder. Also assume that, if an aircraft enters the paragliders' cylinder, it will be within the cylinder at multiple points.

Output

If the aircraft will not enter the cylinder, print the message:

Flight f is safe.

where f is the flight number.

If the aircraft will intersect the cylinder, print the message:

Incoming! Flight f enters at t_0 and exits at t_1.

where f is the flight number, t_0 is the time in seconds when the aircraft will enter the cylinder, and t_1 is the time in seconds when the aircraft will exit the cylinder. Print the times rounded to two decimal places (i.e., the time 0.274 would be printed as 0.27; the time 0.275 would be printed as 0.28). If the aircraft grazes the cylinder (with an error of 10^{-6}), it is considered to have entered and exited.

Example

Input	Output
0.0 0.0 1000.0 0.0 10000.0 2 1200 -5000.0 0.0 0.0 7500.0 5000.0 500.0 2400 -5000.0 0.0 90.0 7500.0 5000.0 500.0	Incoming! Flight 1200 enters at 0.80 and exits at 1.21. Flight 2400 is safe.
-1000.0 1000.0 2000.0 1000.0 10000.0 1 1000 -6000.0 1000.0 0.0 12000.0 4472.0 2000.0	Incoming! Flight 1000 enters at 1.00 and exits at 1.75.

Problem J. Boring Solitaire

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

During quarantine, Stacking-Knightro (SK) made a new card game. SK respects social distancing and, since SK didn't have other people around, the game is a type of solitaire. SK will shuffle a deck of cards. Then, at every step, SK takes the card at the top of the deck. When taking the card, SK will place the card on top of one of the existing piles of cards or will make a new pile with this card as the top. SK can only place a card on top of a pile if the value of the card is not less than the card at the top of the pile. The goal of the game is to minimize the number of piles when the deck is empty.

Stacking-Knightro started getting pretty good "score" before too long, so the game became pretty boring once SK figured out the "trick". SK wants to know how many different starting card arrangements will end up with at most K piles if the game is played optimally at each step, i.e., each card is placed such that it will result in the best outcome.

The Problem:

We have a deck of cards with values 1 through V and a number of suits S, i.e., there are "$(V \times S)!$" arrangements of the cards in the deck (note that "!" refers to the factorial of an integer). Using the game description above and a desired maximum of K piles, you are to determine how many different starting card orders will end up with at most K piles. Since this value can be large, print the output mod $10^9 + 7$.

Input

There is only one input line; it contains three integers: V ($1 \leq V \leq 100$), representing the number of card values (values are 1 through V), S ($1 \leq S \leq 10^4$), representing the number of suits, and K ($1 \leq K \leq V \leq 100$), representing the maximum number of piles allowed.

Output

Print how many different starting card orders will end up with at most K piles.

Example

Input	Output
2 2 1	4
4 3 3	763937568
3 1 3	6

Explanation of the first Example Input/Output:

There are two card values, two suites, and one pile. There are four card arrangements to win the solitaire:

1. 1_{s1} 1_{s2} 2_{s1} 2_{s2}
2. 1_{s1} 1_{s2} 2_{s2} 2_{s1}
3. 1_{s2} 1_{s1} 2_{s1} 2_{s2}
4. 1_{s2} 1_{s1} 2_{s2} 2_{s1}

Problem K. Sum of a Function

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

Everyone knows that Arup loves prime numbers! This is why he teaches the cryptography course at UCF. Recently, Arup defined the following function on positive integers, n, greater than 1:

$$f(n) = \text{ the smallest prime factor of } n$$

For example, $f(14) = 2$, $f(15) = 3$, $f(16) = 2$ and $f(17) = 17$.

Using this function, we can generate a sequence of terms $f(s)$, $f(s+1)$, $f(s+2)$, ..., $f(e)$, where s designates the starting function input and e designates the ending function input.

Arup thinks these sequences are interesting, but what he's really curious about is finding the sum of the k minimum elements in one of these sequences. Can you write a program to help him?

Given s, e, and k, find the sum of the k minimum values in the sequence $f(s)$, $f(s+1)$, $f(s+2)$, ..., $f(e)$.

Input

The first and only input line will contain three positive integers, s ($2 \leq s \leq 10^{18}$), e ($s{+}100 \leq e \leq s{+}10^6$), and k ($1 \leq k \leq 0.9{*}(e{-}s{+}1)$), representing (respectively) the starting function input, the ending function input, and the number of minimum terms to sum.

Output

On a line by itself, print the sum of the k minimum terms of the designated sequence.

Example

Input	Output
100 200 70	165
213 419 169	546

Note: Even though the input specification does not allow "14 17 3" as an input case (i.e., this case will not be in the judge data), it is a simple case that you may want to use for testing purposes – the output (7) can be verified easily on paper. (BTW, the intended solution should solve this case properly anyway.)

Problem L. Trading Cards

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

You've decided to get rid of your Trading Card Game (TCG) collection. The possible financial activities are as follows:

- You can sell individual cards to Bearimy's Card Emporium.
- You can buy individual cards from Bearimy's Card Emporium.
- Card price is the same regardless of the transaction (buy or sell) with the Emporium.
- You can sell a set of cards (called a collection) to your friend Jeremy. Jeremy enjoys showing off different collection sets, so Jeremy will buy multiple sets but only one of each set. You have to, of course, have the cards in a set to be able to sell such set to Jeremy.
- If a card is needed in different collection sets, you only need one copy of that card (and not multiple copies of that card) to form all those sets to sell to Jeremy; you need the other cards in the sets as well. For example, if cards $\{1,\ 2\}$ is a set, cards $\{2,\ 3\}$ is a set, and cards $\{1,\ 3,\ 4\}$ is a set, to sell these three sets to Jeremy, you only need one of each card 1, 2, 3, and 4.
- Jeremy only buys collection sets and not individual cards.
- A collection set is not necessarily more/less expensive than the sum of its component cards.

Given a list of cards, including their card shop cost and whether you own them, and a list of collection sets and the value Jeremy will pay for those sets, determine the maximum amount of money you can earn. This is done by buying from (and/or selling to) the card shop and selling the sets to Jeremy. Note that you choose what to buy from (and/or sell to) the card shop and what to sell to Jeremy. Your earning will be:

$$(\text{cards sold to the shop}) + (\text{sets sold to Jeremy}) - (\text{cards bought from the shop})$$

Input

The first input line contains an integer, n ($1 \leq n \leq 50$), representing the number of cards. Each of the next n input lines contains two integers: v_i ($1 \leq v_i \leq 10^4$), representing the Bearimy card value, and h_i ($0 \leq h_i \leq 1$), representing whether you currently own the card ($h_i = 1$) or not ($h_i = 0$). The cards are provided in the order of indices (1-indexed).

Following the individual card specification will be the description for the collection sets. The first input line in this section contains an integer, m ($1 \leq m \leq 50$), representing the number of sets. The set descriptions follow, each set consisting of two consecutive input lines. The first line of each set description contains two integers: c_j ($1 \leq c_j \leq n$), representing the number of cards in the set, and w_j ($1 \leq w_j \leq 10^4$), representing the value of the set. The second input line of each set description contains c_j distinct positive integers between 1 and n; these values represent the indices (1-indexed) of the cards that belong to the set.

Output

Print a single integer, P, representing the maximum profit that can be earned by buying and selling cards with Bearimy and selling collection sets to Jeremy.

Example

Input	Output
3 2 0 13 1 15 1 1 3 6 1 2 3	28
3 2 0 13 1 15 1 1 3 600 1 2 3	598
4 7 1 3 0 2 1 1 0 3 4 4 1 2 3 4 2 4 2 3 2 4 3 4	11