Problem A. Sharing Birthdays

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

In a room of 23 people, there is a 50-50 chance of at least two people having the same birthday; in a room of 75, there is a 99.9% chance of at least two people having the same birthday.

Given a set of birthdays (each in the form of `mm/dd`), determine how many different birthdays there are, i.e., duplicates should count as one.

Input

The first input line contains an integer, n ($1 \leq n \leq 50$), indicating the number of birthdays. Each of the next n input lines contains a birthday in the form of `mm/dd`. Assume `mm` will be between 01 and 12 (inclusive) and `dd` will be between 01 and 31 (inclusive). Also assume that these values will be valid, e.g., there will not be `02/31` in the input. (Consider `02/28` and `02/29` as different days even though people born on `02/29` usually celebrate their birthdays on `02/28`.)

Output

Print how many different birthdays there are.

Example

Input	Output
3 07/09 10/14 07/09	2
7 10/20 11/22 10/20 10/22 11/20 10/20 11/22	4

Problem B. Digit Count

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

There are many ways to count the frequencies of letters but can they be applied to digits?

Given a range (in the form of two integers) and a digit (0-9), you are to count how many occurrences of the digit there are in the given range.

Input

There is only one input line; it provides the range and the digit. Each integer for the range will be between 1000 and 9999 (inclusive) and the digit will be between 0 and 9 (inclusive). Assume the first integer for the range is not greater than the second integer for the range.

Output

Print the number of occurrences of the digit in the given range.

Example

Input	Output
1000 1000 0	3
1000 1001 0	5
8996 9004 5	0
9800 9900 5	20

Problem C. Tetrooj Box

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

Dr. Orooji's children have played Tetris but are not willing to help Dr. O with a related problem. Dr. O's children don't realize that Dr. O is lucky to have access to 100+ great problem solvers and great programmers today!

Dr. O knows the length of the base for a 2D box and wants to figure out the needed height for the box. Dr. O will drop some 2D blocks (rectangles) on the base. A block will go down until it lands on the base or is stopped by an already-dropped block (i.e., it lands on that block). After all the blocks have been dropped, we can determine the needed height for the box – the tallest column is the needed height (please see pictures on the next page corresponding to Example Input/Output).

Input

The first input line contains two integers: b ($1 \leq b \leq 100$), indicating the length of the base and r ($1 \leq r \leq 50$), indicating the number of blocks (rectangular pieces) to be dropped. Each of the next r input lines contains three integers: a block's horizontal length h ($1 \leq h \leq 100$), the block's vertical length v ($1 \leq v \leq 100$), and c ($c \geq 1$), the leftmost column the block is dropped into. Assume that the h and c values will be such that the block will not go beyond the box base, i.e., $(c + h - 1) \leq b$.

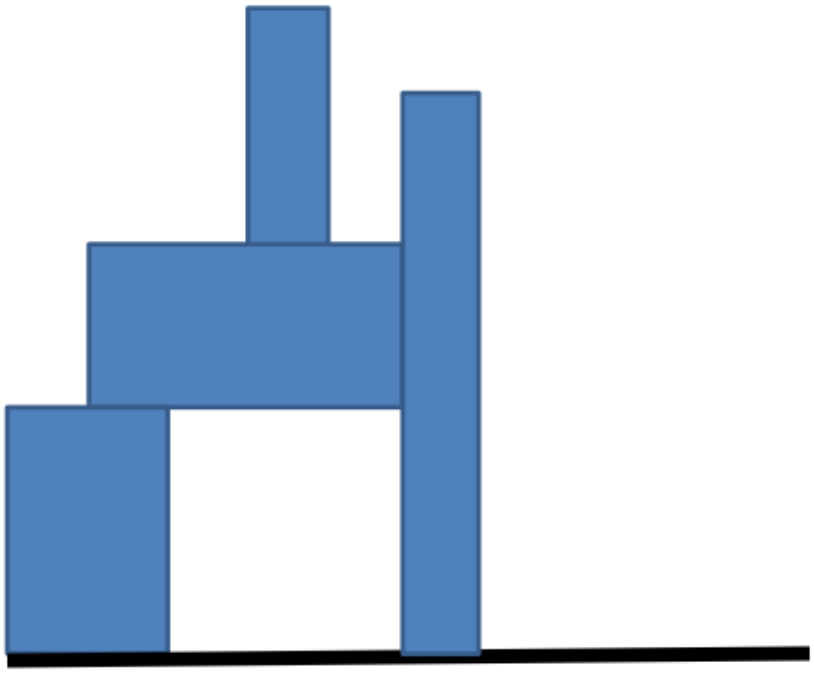
Output

Print the needed height for the box (the tallest column is the height).

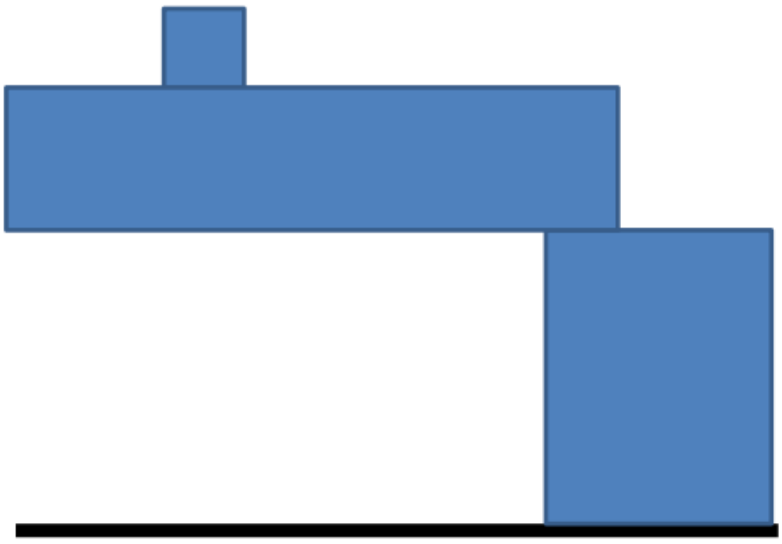
Example

Input	Output
10 4 2 3 1 4 2 2 1 7 6 1 3 4	8
10 3 3 4 8 8 2 1 1 1 3	7

First Sample Input/Output:



Second Sample Input/Output:



Problem D. Judging Assistant for Contest

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

When people try competitive programming for the first time, it can be challenging for some to write code according to the constraints of the contest, even if they are already good coders! Of course, the best way to learn is by doing, and that is why many contests have a "practice" or warmup session, which includes testing out the full process of submitting code and getting responses from the judges. Still, there are always some people who do not attend that session and then make time-wasting mistakes in the real contest, where it counts! For example, they print out prompts for inputs, or forget to return zero from their C program. This happens often enough that the judges need some programs to help them evaluate the submissions.

Given information about a programming contest problem and a submission for that problem, help the judges determine the best response.

Input

The first input line contains the designated "filename" for the contest problem which is a string of 1 to 20 lowercase letters. Recall that this filename does not include the extension (.c, .cpp, .java, .py).

The second input line contains the name of the submitted file, a string of 1 to 70 characters. This filename may include an extension, though the contestant might have used an invalid extension (e.g., .html, .pl, .rb, .asp, ...). Note that this file name consists of characters and not just letters, e.g., the file name may be "`C:\My Documents\graph.py`".

The third input line contains three single-space-separated integers: r ($0 \leq r \leq 10$), indicating the return code of the submitted program, d ($1 \leq d \leq 10$), indicating the time limit in seconds allowed for a correct program to run, and e ($0 \leq e \leq 20$), indicating the elapsed time in seconds while the program was running. (Note that, in a real contest, it may not be possible to run a submitted program but that aspect is not included in this problem to simplify the problem.)

The fourth input line contains an integer, c ($1 \leq c \leq 10$), indicating the number of output lines produced by a correct program. The following c input lines provide the correct output; each line will contain zero to 70 characters and the first and last line(s) will contain at least one non-blank character.

The next input line contains an integer, t ($0 \leq t \leq 10$), indicating the number of output lines produced by the submitted program. The following t input lines provide the output produced by the submitted program; each line will contain zero to 70 characters. Note that, unlike the correct output, it is not guaranteed that the first and last line(s) of the submitted output will contain at least one non-blank character, i.e., the submitted output could be all blanks. Note also that, in a real contest, a submitted program may have far more output lines than expected and some output lines may far exceed the expected length but those aspects are not included in this problem to simplify the problem.

Output

If the submitted file name does not match (case-sensitively) the designated problem filename, or if it doesn't have one of the valid extensions (.c, .cpp, .java, or .py), print the message "`Compile Error`". Otherwise, if the return code of the program is not zero, print "`Run-Time Error`". Otherwise, if the elapsed time for the submitted program exceeds the time limit, print "`Time Limit Exceeded`". Otherwise, if the submitted program output does not match the correct output (line by line and character by character), print "`Wrong Answer`". Otherwise, print "`Correct`".

Example

Input	Output
triangle MyTriangle.py 0 3 0 1 scalene 0	Compile Error
graph graph.py 5 3 3 1 done 1 done	Run-Time Error
dust dust.c 0 10 11 3 12cm of dust 3cm of dust Impossible 1 Please enter shelf size:	Time Limit Exceeded
awesome awesome.java 0 5 5 1 Result = 100 2 Result = 100	Wrong Answer
awesome awesome.java 0 5 5 2 Result = 100 2 RESULT = 100	Wrong Answer
awesome awesome.java 0 5 5 1 Everything is awesome! 1 Everything is awesome!	Correct

Problem E. Thirsty Professors

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

Dr. Orooji and Dr. Meade were lost in a desert and they were extremely thirsty. They each had a stick so they decided that they can form a shape (e.g., "V" shape, "X" shape, etc.) facing the skies and, when it rains, water would collect in the top part of the shape and then they can drink it; please see the picture on the next page.

Making the shape would be easier if one person holds both sticks but neither professor was willing to give up his stick. So, they tried to form the shape together, each holding one stick. And, we know how coordinated the professors can be!

Given two line segments, the first line with positive slope and the second line with negative slope, compute the area for water collection. When checking for intersecting (touching), if two points are within 10^{-6} of each other, consider the points the same.

Input

There are two input lines, each describing a line segment. The first input line contains four integers ($0 < x_1,\ y_1,\ x_2,\ y_2 < 10^3$; $x_2 > x_1$ and $y_2 > y_1$), describing the first stick. The second input line contains four integers ($0 < x_3,\ y_3,\ x_4,\ y_4 < 10^3$; $x_4 < x_3$ and $y_4 > y_3$), describing the second stick. Note that neither line segment will be parallel to x-axis or parallel to y-axis.

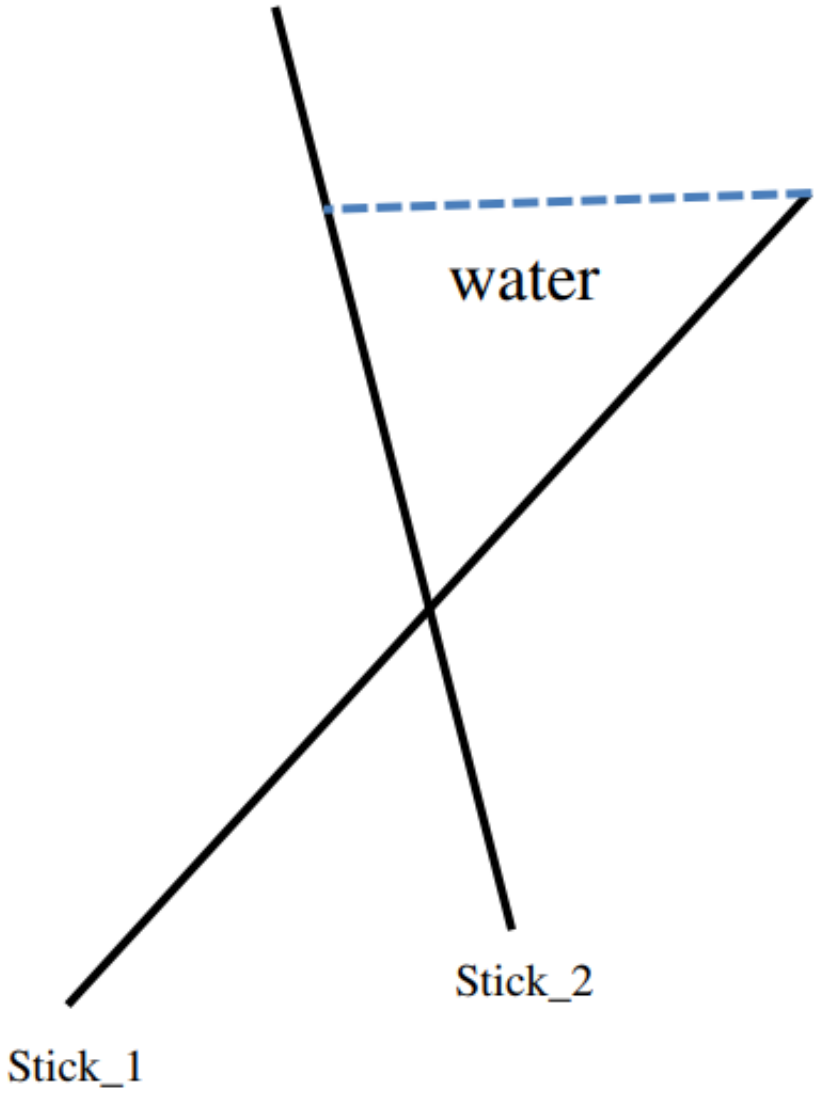
Output

Print the area for water collection. Your output may have any number of digits after the decimal point; answers within 10^{-5} absolute of the judge output will be considered correct.

Note that, as illustrated in the second Example Input/Output, if the two line segments do not intersect, it will not be possible for the water to collect and, as such, the output will be zero.

Example

Input	Output
5 2 9 6 5 2 3 4	4.0
11 14 18 15 13 18 10 20	0.0
5 2 9 6 12 1 6 10	0.03333


water
Stick_2
Stick_1

Problem F. Hidden Message

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

John was reading the local newspaper, and noticed that the phrase "chime a cork teen" could be split into three sub-phrases "eat", "more", and "chicken". Note that the three sub-phrases combined contain exactly the same letters as the original phrase and the letters in each sub-phrase appear in the same order as they appear in the original phrase. Note also that the number of occurrences of each letter in the three sub-phrases combined is the same as that of the original phrase.

John began to theorize that the newspapers were sending him messages, but you decide to show him that a message like that was not abnormal. You want to determine the number of ways a phrase can be broken down into three words that John finds.

Given three sub-phrases and the original phrase, determine the number of ways the sub-phrases can be formed from the original phrase. The number of ways can be quite large, so determine the number modulo 1,000,000,007.

Input

The input consists of four lines. Each of the first three input lines contains 1-100 lowercase letters, representing a sub-phrase. The fourth input line contains 3-300 lowercase letters, representing the original phrase. Note that the sum of the lengths of the three sub-phrases is equal to the length of the original phrase.

Output

Print a single integer representing the number of ways to partition the original phrase into three groups where each group is one of the three sub-phrases. Print the count modulo 1,000,000,007.

Example

Input	Output
eat more chicken chimeacorkteen	2
the great depression depressigortheneat	2
a a a aaa	6

Problem G. Make the Team

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

You are eager to make the programming team. You have decided that if you watch several videos, you will increase your chances of making the team. Each video you want to watch is one hour long, and each video plays at specific times. Naturally, you want to get through all the videos as fast as possible, to leave more time to practice!

For example, if you want to watch two videos and the first one is available at the time intervals $[1,\ 2)$, $[4,\ 5)$, $[8,\ 9)$ and $[12,\ 13)$, and the second video is available at the time intervals $[4,\ 5)$, $[7,\ 8)$, and $[11,\ 12)$, then the earliest time at which you can complete the two videos is time 5. You can accomplish this by watching the first video in the time interval $[1,\ 2)$ and the second one at the time interval $[4,\ 5)$. Note that all videos play for precisely the length of one time interval, and one can watch back to back videos. Thus, if one video plays at the interval $[x,\ x+1)$ and another video plays at the interval $[x+1,\ x+2)$, where x is a positive integer, both can be watched, back to back.

Given a list of times that each video you want to watch is available, determine the earliest time at which you can complete watching all of the videos. Note that the videos can be watched in any order as long as the time intervals allow.

Input

The first input line contains a single integer, n ($1 \leq n \leq 200$), indicating the number of videos you would like to watch. Each of the next n input lines describes a video you want to watch. The i^{th} of these input lines starts with an integer, t_i ($1 \leq t_i \leq 30$), representing the number of times video i is available to watch. This is followed by t_i space separated values indicating the starting time video i is available to watch. The list of times for each video will be a strictly increasing list of positive integers, with a maximum value of 1000. It is guaranteed that there will be at least one arrangement that allows you to watch all of the videos.

Output

Print the earliest time at which you can complete watching all of the videos.

Example

Input	Output
3 2 4 6 3 4 9 11 1 4	10
4 2 3 11 3 2 9 11 2 2 3 2 3 9	12

Problem H. Decoder Ring

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

Cereal Companies usually include toys in their boxes to attract kids to their products. One of the toys in the 1990's was as follows: the toy had a piece of paper showing a long string of letters (we will refer to this string as the ciphertext). The toy also had a list of positive integers, which we will refer to as the key. The first integer of the key was the distance from the beginning of the ciphertext to get you to the first letter of a plaintext message, i.e., the first integer would provide how far to advance in the ciphertext to arrive at the first letter of the plaintext message. The second integer of the key was the distance from the first letter in the ciphertext to get you to the second letter of the plaintext message. The third integer of the key was the distance from the second letter in the ciphertext to get you to the third letter in the plaintext message, and so on. Taking the steps provided in the key would reveal the plaintext message. The sum of the integers in the key would not, of course, exceed the length of the ciphertext. For example, if the ciphertext was "`abcdoefgholijk`" and the key was {3, 2, 5, 1}, the *plaintext* message would be "`cool`".

But, the kids today have access to several computing devices so the above problem would be solved in one millisecond by the kids! The Unlimited Cereal Factory has modified the above toy. The new version provides a string and an integer K; the ciphertext is created by repeating (concatenating) the given string K times. The key is not provided either; rather the plaintext message is provided and the challenge is to determine how many different keys could be selected to extract the plaintext message from the ciphertext. Again, the sum of integers cannot exceed the length of the ciphertext.

Let's use the first Example Input/Output to explain the problem further. The ciphertext is "`abcde`" repeated 4 times so the ciphertext is "`abcdeabcdeabcdeabcde`", and the plaintext message is "`abc`". The plaintext message can be extracted from the ciphertext with 20 different keys, each key (list of integers) providing the distances. Some of the 20 possible keys that can be used to extract the plaintext "`abc`" from the ciphertext are {1, 1, 1}, {1, 1, 6}, {1, 1, 11}, {1, 1, 16}, and {1, 6, 1}.

Given a ciphertext formed by a string repeated a constant number of times, and a desired plaintext message, determine the number of ways you can create the plaintext message represented by positive offsets on the ciphertext. Since the number of ways can be quite large, output the answer modulo 1,000,000,007.

Input

The first input line contains a string ($1 \leq length \leq 100$), starting in column 1 and consisting of only lowercase letters. The second input line contains an integer, k ($1 \leq k \leq 10^{18}$), representing the number of times the string is repeated to derive the ciphertext. The third input line contains the plaintext message ($1 \leq length \leq 50$), starting in column 1 and consisting of only lowercase letters.

Output

Print a single integer, representing the number of ways to form the plaintext message from the ciphertext. Again, the sum of the integers in the list cannot, of course, exceed the length of the ciphertext.

Example

Input	Output
abcde 4 abc	20
taco 25 tacocat	1184040

Problem I. Wedding DJ

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

You have always wanted to be a DJ and finally got your first opportunity! You have a list of songs that you will play at a wedding in the order the songs appear in the list. Each song has a "*fun level*" but the problem is that B&G (the bride and the groom) do not want any song to be played after a song with a lower fun level, i.e., B&G consider the wedding playlist good only if the fun level of the songs do not ever decrease. Fortunately, you can adjust the fun level of songs. You can choose to change all songs with fun level x to fun level y, e.g., you can choose to change all songs with fun level 10 to fun level 7 (or to fun level 18). Note that:

- When you choose to change all songs with fun level x to fun level y, even though all the songs with fun level x are changed, this is considered as one change.
- When you choose to change all songs with fun level x, all the songs with fun level x change and not a selected subset of the songs with fun level x.
- If you change all songs with fun level x to fun level y and then you decide to change all the songs with fun level y to fun level z, the second change applies to the updated list and not the original list. For example, if the original list is $\{\ldots\ 3\ldots\ 8\ldots\ 3\ldots\ 8\ldots\ 3\ldots\}$ and you decide to change fun level 3 to 8 and then decide to change fun level 8 to 2, five songs change their fun level to 2 and not two songs.

Given the order of the fun level of a list of songs, determine the minimum number of playlist adjustments in the form of transforming all songs of level x into level y, such that the fun level of the songs of the playlist is non-decreasing.

Input

The first input line contains an integer, n ($1 \leq n \leq 10^5$), representing the number of songs in the playlist. The following input line contains n space separated integers, representing the fun level for the songs in the order they appear in the playlist; each fun level is between 1 and 10^9 (inclusive).

Output

Print the minimum number of adjustments to make the playlist's fun level non-decreasing.

Example

Input	Output
10 1 7 1 7 8 5 8 3 8 8	3
6 1 6 3 4 2 1	4

Explanation

Explanation of the first Example Input/Output:

One way to make the list non-decreasing is:

- Change all 7's to 1

- Change all 8's to 3
- Change all 5's to 3.

for a total of 3 changes.

Problem J. Median Inversion String

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

This problem uses two concepts: median and inversion.

Median: Assume we have a sorted list of elements e_1, e_2, e_3, ... We define median as the element in the middle of the list (i.e., halfway through the list). If there are an odd number of elements in the list, there is only one median element, e.g., if the list has 13 elements, then e_7 is the median element (there are as many elements in the list before e_7 as there are after e_7). If there are an even number of elements in the list, there are two median elements, e.g., if the list has 14 elements, then e_7 and e_8 are the median elements (there are as many elements in the list before e_7 as there are after e_8).

Inversion: Assume we have a string containing only the letters '`A`' and '`B`', e.g., `AAABABAA`. We define an inversion to be a pair of letters in the string where one letter is `B`, the other letter is `A`, and the position of `B` in the string is earlier than the position of `A` in the string. For our sample string, the `B` at position 4 and the `A` at position 5 form an inversion. Similarly, the `B` at position 4 and the `A` at position 7 form an inversion. There is a total of five inversions in the sample string.

Your friend, Michio, has a list of strings, each string containing only the letters '`A`' and '`B`'. Each string has n letters. Each string Michio has contains exactly k inversions. Michio claims that he has all strings of exactly k inversions. You want to test his theory.

Michio has sorted the strings in lexicographical order, and you will give the median string(s) to check if he has the correct strings.

Given the length (number of letters) in each string and the desired number of inversions, output the median string(s) in the list of strings with the given length.

Input

There is only one input line; it consists of two integers: n ($1 \leq n \leq 60$), representing the length of each string in the list Michio has, and k, representing the number of inversions per string. It is guaranteed that k will be selected such that at least one string containing `A`'s and `B`'s of length n will have exactly k inversions. (Note that the bounds for k are implied by the other constraints in the problem, i.e., the bounds for k need not be provided.)

Output

If the number of unique strings of `A`'s and `B`'s of length n with k inversions is odd, print a single line with the median string. If the number of unique strings of `A`'s and `B`'s of length n with k inversions is even, then print two lines each containing one of the median strings (these two strings should be printed in lexicographical order).

Example

Input	Output
`10 3`	`AAABABABBB`
`3 0`	`AAB` `ABB`

Problem K. Check List

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

When test driving a vehicle, you are asked to make a checkmark on a tablet. Upon inspection, you have noticed many fingerprints on the screen. Aside from the rush of disgust, you notice that the fingerprints on the screen can be represented by a series of 2D points on a standard Cartesian grid. You know that checkmarks can be uniquely defined by three points; these three points have distinct x coordinates and the three points also have distinct y coordinates. The leftmost point must be lower than the rightmost point and higher than the middle point. Now you want to know how many unique checkmarks you can make using the 2D points.

Let's consider some examples. The three points $(1,\ 2)$, $(2,\ 1)$, and $(3,\ 2)$ do not form a valid checkmark because the leftmost and rightmost points are at the same height. The three points $(1,\ 2)$, $(2,\ 1)$, and $(2,\ 3)$ do not form a valid checkmark because two points have the same x coordinates. The three points $(1,\ 2)$, $(3,\ 1)$, and $(4,\ 9)$ do form a valid checkmark.

Given a list of 2D points, determine the number of unique checkmarks that can be formed from them.

Input

The first input line contains a single integer, n ($1 \leq n \leq 10^5$), representing the number of points. Each of the following n input lines contains two integers, x_i and y_i ($1 \leq x_i,\ y_i \leq 10^9$), representing the location of a point on the 2D grid. No two points will be identical.

Output

Print the number of distinct checkmarks that can be formed by the given 2D points.

Example

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