acm International Collegiate Programming Contest

IBM event sponsor

2016 ACM ICPC
Southeast USA Regional
Programming Contest
Division 1

Hosted by:

College of Charleston

Florida Institute of Technology

Kennesaw State University

University of West Florida

Alphabet

A string of lowercase letters is called *alphabetical* if some of the letters can be deleted so that the only letters that remain are the letters from `**a**' to `**z**' in order.

Given a string ***s***, determine the minimum number of letters to add anywhere in the string to make it *alphabetical*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The only line of input contains a string ***s*** (**1**≤|***s***|≤**50**) which contains only lowercase letters.

Output

Output a single integer, which is the smallest number of letters needed to add to ***s*** to make it alphabetical.

Sample Input	Sample Output
xyzabcdefghijklmnopqrstuvw	3
aiemckgobjfndlhp	20

Base Sums

Given three values ***n***, ***a***, and ***b***, find the smallest ***m***>***n*** such that the sum of the digits of ***m*** in base ***a*** is the same as the sum of digits of ***m*** in base ***b***.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. There will be a single line of input, with three integers, ***n*** (**0**≤***n***≤**10**^{16}), ***a*** and ***b*** (**2**≤***a***<***b***≤**36**), all of which will be in base **10**.

Output

Output a single integer, ***m***, which is the smallest number greater than ***n*** such that the sum of its digits in base ***a*** is the same as the sum of its digits in base ***b***. Output ***m*** in base **10**.

Sample Input	Sample Output
66 10 16	144
24 4 15	90
9358385 11 32	9437362

Buggy Robot

There is a robot in a 2D grid. The grid consists of empty cells and obstacles, and there is exactly one cell that is the exit. The robot will exit the grid if it ever reaches the exit cell. Empty cells are denoted as '`.`', the robot's initial position is denoted as '`R`', obstacles are denoted as '`#`', and the exit is denoted as '`E`'.

You can program the robot by sending it a series of commands. The series of commands is a string consisting of characters: '`L`' (move one square left), '`U`' (move one square up), '`R`' (move one square right), or '`D`' (move one square down). If the robot would run into an obstacle or off the edge of the grid, it will ignore the command (but it will continue on to future commands, if there are any).

Your friend sent a series of commands to the robot, but unfortunately, the commands do not necessarily take the robot to the exit.

You would like to fix the string so that the robot will reach the exit square (note once the robot reaches the exit, it stops, even if there are more commands in the string). You can fix the string with a sequence of operations. There are two operations: inserting a command anywhere in the string, or deleting a command anywhere in the string. What is the minimum number of operations you would need to fix the program?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***r*** and ***c*** (**2**≤***r***,***c***≤**50**) which are the number of rows and number of columns of the grid. The next ***r*** lines will each contain a string with exactly ***c*** characters. Each character is one of '`.`' (Empty), '`R`' (the Robot), '`#`' (an Obstacle), or '`E`' (the Exit). This is the grid. There will be exactly one '`R`' and one '`E`' in the grid, and it will always be possible to navigate the robot to the exit. The last line of input will contain a string ***s*** (**1**≤|***s***|≤**50**) of commands. The string ***s*** will consist only of characters '`L`' (left), '`R`' (right), '`U`' (up) or '`D`' (down).

Output

Output a single integer, which is the minimum number of operations necessary to fix the command string so that the robot makes it to the exit.

Sample Input	Sample Output
3 3 R.. .#. ..E LRDD	1
2 4 R.#. #..E RRUUDDRRUUUU	0

Enclosure

In the Dark Forest, you control some trees. The territory you control is defined by the smallest convex shape that contains all of the trees that you control. Your *power* is defined by the area of this convex shape. You may control trees inside the convex shape that are not on the edge of the shape.

You currently control ***k*** trees of the ***n*** in the forest. You want to extend your power by gaining control over a single additional tree, somewhere in the forest. After acquiring the single tree that most increases your power, what is the area of your new shape?

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two space-separated integers ***n*** and ***k*** (**3≤*k*<*n*≤100,000**), where ***n*** is the total number of trees, and ***k*** is the number of trees that you control.

The next ***n*** lines each have two space-separated integers ***x*** and ***y*** (**$-10^9 \le x,y \le 10^9$**) specifying the locations of the ***n*** trees. The first ***k*** trees in the list are the trees that you control. No three trees will have collinear locations. Note that there may be trees that you don't control within your shape.

Output

Output a single floating point number, which is the largest area you can achieve by controlling a single additional tree. Output this number to exactly one decimal place.

Sample Input	Sample Output
5 3 -5 -5 -5 5 5 -5 -4 6 5 5	100.0

Illumination

Consider a square grid with lamps in fixed locations. Each lamp can either illuminate its row or its column, but not both. The illumination of each lamp extends over a limited distance.

Any square in the grid should only be illuminated by at most one lamp in its row and by at most one lamp in its column (one of each is acceptable, as is just the row, just the column, or neither). Determine if it is possible for all lamps to be lit while satisfying these constraints.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains three positive integers, ***n***, ***r*** and ***k*** (**1**≤***n***,***r***,***k***≤**1,000**, ***k***≤***n***x***n***), where ***n*** is the size of one side of the square grid, ***r*** is the maximum reach of a lamp, and ***k*** is the number of lamps. The next ***k*** lines will each contain two positive integers ***i*** and ***j*** (**1**≤***i***,***j***≤**n**), indicating that there is a lamp in the grid at row ***i***, column ***j***.

Each lamp can illuminate squares at most ***r*** units away, and can also illuminate its own square, so the maximum number of squares it can illuminate is **2*r*+1**. All lamps will be in distinct locations.

Output

Output a single integer, **1** if it is possible to light all of the lamps and **0** if it is not possible.

Sample Input	Sample Output
3 2 5 1 1 1 3 3 1 3 3 2 2	1
3 2 6 1 1 1 2 1 3 3 1 3 2 3 3	0

InTents

A circus is constructing their tent. The tent is a large piece of canvas held up by a given number of various length tent poles. The tent poles go in specific places on the ground, but which tent pole goes in which place is up to you. You need to choose a placement for the given tent poles that maximizes the total volume under the tent.

There will always be one central pole at the origin; the other poles are distributed around the periphery. The tent is always drawn tight between the central pole and two adjacent poles on the periphery, forming a perfect triangle. Only the volume under these triangles formed by two adjacent outer poles and the central origin pole counts towards the total volume. Adjacency is by angle around the origin.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains an integer ***n*** (**3**≤***n***≤**30**), which is the number of poles.

The next ***n***-1 lines each contains two integers ***x*** and ***y*** (**-1,000**≤***x***,***y***≤**1,000**), representing a 2D coordinate, giving the locations where the poles may be placed. The locations may not be in order around the origin. After that, there will be ***n*** lines, each containing a single integer ***h*** (**1**≤***h***≤**100**). These are the heights of the poles.

One pole must be placed at the origin, and the rest must be placed at the (***x***,***y***) coordinates in the input. The (***x***,***y***) locations will surround the origin; that is, the polygon formed by the (***x***,***y***) locations, in order (by angle around the origin), will strictly include the origin. No two holes will be at the same angle with the origin (i.e. no triangle of roof fabric will have area **0**).

Output

Output a single floating point number, which is the maximum volume achievable under the tent. Output this number to exactly two decimal places, rounded.

Sample Input	Sample Output
5 100 100 -200 -200 300 -300 -400 400 30 20 50 60 10	8566666.67

Islands

You are mapping a faraway planet using a satellite. The planet's surface can be modeled as a grid. The satellite has captured an image of the surface. Each grid square is either land (denoted as '**L**'), water (denoted as '**W**'), or covered by clouds (denoted as '**C**'). Clouds mean that the surface could either be land or water; you cannot tell.

An island is a region of land where every grid cell in the island is connected to every other by some path, and every leg of the path only goes up, down, left or right.

Given an image, determine the minimum number of islands that is consistent with the given image.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers, ***r*** and ***c*** (**1**≤***r***,***c***≤**50**), which are the number of rows and the number of columns of the image. The next ***r*** lines will each contain exactly ***c*** characters, consisting only of '**L**' (representing *Land*), '**W**' (representing *Water*), and '**C**' (representing *Clouds*).

Output

Output a single integer, which is the minimum number of islands possible.

Sample Input	Sample Output
4 5 CCCCC CCCCC CCCCC CCCCC	0
3 2 LW CC WL	1

Paint

You are painting a picket fence with ***n*** slats, numbered from **1** to ***n***. There are ***k*** painters willing to paint a specific portion of the fence. However, they don't like each other, and each painter will only paint their given portion of the fence if no other painter overlaps their portion.

You want to select a subset of painters that do not conflict with each other, in order to minimize the number of unpainted slats. For example, suppose there are **8** slats, and **3** painters. One painter wants to paint slats **1→3**, one wants to paint **2→6**, and one wants to paint **5→8**. By choosing the first and last painters, you can paint most of the slats, leaving only a single slat (slat **4**) unpainted, with no overlap between painters.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains two integers ***n*** (**1**≤***n***≤10^{18}) and ***k*** (**1**≤***k***≤**200,000**), where ***n*** is the number of slats and ***k*** is the number of painters. Each of the next ***k*** lines contains two integers ***a*** and ***b*** (***1***≤***a***≤***b***≤***n***), indicating that this painter wants to paint all of the slats between ***a*** and ***b***, inclusive.

Output

Output a single integer, which is the smallest number of slats that go unpainted with an optimal selection of painters.

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

Periodic Strings

Define a *k-periodic* string as follows:

A string ***s*** is *k-periodic* if the length of the string $|s|$ is a multiple of ***k***, and if you chop the string up into $|s|/k$ substrings of length ***k***, then each of those substrings (except the first) is the same as the previous substring, but with its last character moved to the front.

For example, the following string is *3-periodic*:

abccabbcaabc

The above string can break up into substrings **abc**, **cab**, **bca**, and **abc**, and each substring (except the first) is a right-rotation of the previous substring (**abc→cab →bca →abc**).

Given a string, determine the smallest ***k*** for which the string is *k-periodic*.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The single line of input contains a string ***s*** (**1**≤$|s|$≤**100**) consisting only of lowercase letters.

Output

Output the integer ***k***, which is the smallest ***k*** for which the input string is *k-periodic*.

Sample Input	Sample Output
aaaaaaaa	1
abbaabbaabba	2
abcdef	6
abccabbcaabc	3

Water

A water company is trying to provide water from its pumping station to a mansion. The company owns ***n*** water stations, numbered **1**..***n***, which are connected by a variety of pipes. Water can flow through both directions of a pipe, but the total amount of water that can flow through the pipe is bounded by the capacity of the pipe.

The water company is constantly improving the pipes, increasing the capacity of various pipes. The water company is conducting ***k*** improvements (each of which is permanent after it is executed). An improvement consists of taking a pipe between two locations and increasing its capacity by a fixed amount, or installing a pipe between two locations which are not directly connected by a pipe.

After each improvement, the water company wants to know the maximum amount of water the mansion could receive.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains three integers, ***n*** (**2**≤***n***≤**100**), ***p*** (**0**≤***p***≤ $\frac{n(n-1)}{2}$), and ***k*** (**1**≤***k***≤**10,000**), where ***n*** is the number of stations, ***p*** is the number of initial pipes, and ***k*** is the number of improvements. The first station in the list is always the pumping station, and the second is always the mansion.

The next ***p*** lines will describe the pipes in the initial setup. The lines will each contain three integers, ***a***, ***b*** (**1**≤***a***<***b***≤***n***) and ***c*** (**1**≤***c***≤**1,000**), which indicates that stations ***a*** and ***b*** are connected by a pipe with capacity ***c***. No (***a***,***b***) pair will appear more than once in this section.

The next ***k*** lines will describe the improvements. The lines will each contain three integers, ***a***, ***b*** (**1**≤***a***<***b***≤***n***) and ***c*** (**1**≤***c***≤**1,000**), which indicates that the pipe connecting stations ***a*** and ***b*** has its capacity increased by ***c*** (if there is currently no pipe between ***a*** and ***b***, then one is created with capacity ***c***). Note that it is possible for an (***a***,***b***) pair to be repeated in this section.

Output

Output ***k***+**1** integers, each on its own line, describing the maximum amount of water that can reach the mansion. The first number is the amount of water reaching the mansion in the initial configuration. The next ***k*** numbers are the amounts of water reaching the mansion after each improvement.

Sample Input	Sample Output
3 2 1 1 3 10 2 3 1 2 3 15	1 10
6 10 2 1 3 2 1 4 6 1 5 1 3 5 8 4 5 7 2 4 3 2 5 4 2 6 1 5 6 9 3 6 5 2 6 9 1 6 3	8 9 12

Zigzag

A sequence of integers is said to *Zigzag* if adjacent elements alternate between strictly increasing and strictly decreasing. Note that the sequence may start by either increasing or decreasing. Given a sequence of integers, determine the length of the longest subsequence that *Zigzags*. For example, consider this sequence:

1 2 3 4 2

There are several *Zigzagging* subsequences of length **3**:

1 3 2 **1 4 2** **2 3 2** **2 4 2** **3 4 2**

But there are none of length greater than **3**, so the answer is **3**.

Input

Each input will consist of a single test case. Note that your program may be run multiple times on different inputs. The first line of input contains an integer ***n*** (**1≤*n*≤1,000,000**) which is the number of integers in the list. Each of the following ***n*** lines will have an integer ***k*** (**1≤*k*≤1,000,000**)

Output

Output a single integer, which is the length of the longest *Zigzagging* subsequence of the input list.

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