**Access Changer**

You are converting old code for a new compiler version. Each "->" string should be replaced by ".", but this replacement shouldn't be done inside comments. A comment is a string that starts with "//" and terminates at the end of the line.

You will be given a sequence of lines; a ***program***, containing the old code. Printa a sequence of lines containing the converted version of the code.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with an integer ***N***, ***1<=N<=50*** and a program in the next N lines. You can assume the following conditions will always be true;

* Each element of program will contain between 1 and 50 characters, inclusive.
* Each character in program will have ASCII value between 32 and 127, inclusive.

**Output Specification**

For each test case, print the converted program, the answer, follow the format on the output example.

**Input Example**

4

Test\* t = new Test();

t->a = 1;

t->b = 2;

t->go(); // a=1, b=2 --> a=2, b=3

3

---> // the arrow --->

---

> // the parted arrow

1

->-> // two successive arrows ->->

**Output Example**

Test\* t = new Test();

t.a = 1;

t.b = 2;

t.go(); // a=1, b=2 --> a=2, b=3

--. // the arrow --->

---

> // the parted arrow

.. // two successive arrows ->->

**Bad Vocabulary**

Little Teddy and Little Tracy are now learning how to speak words. Their mother, of course, doesn't want them to speak bad words. According to her definition, a word W is bad if at least one of the following conditions hold (see the notes in input specification section for definitions):

* W contains the string ***badPrefix*** as a prefix.
* W contains the string ***badSuffix*** as a suffix.
* W contains the string ***badSubstring*** as a contiguous substring that is neither a prefix nor a suffix of W.

You are given a ***vocabulary*** representing the words that Teddy and Tracy are going to learn. Find and print the number of bad words in ***vocabulary***.

**Input Specification**

The input will consist in several test cases; each test case will consist of three lines, the first line will contain ***badPrefix***, ***badSuffix*** and ***badSubstring***, separated by single spaces and with out leading or trailing spaces, the second line of each test case will contain the vocabulary, separated by single spaces and with out leading or trailing spaces. You can assume the following conditions will always be true:

* A prefix of a string is obtained by removing zero or more contiguous characters from the end of the string.
* A suffix of a string is obtained by removing zero or more contiguous characters from the beginning of the string.
* ***badPrefix***, ***badSuffix***, and ***badSubstring*** will each contain between 1 and 50 characters, inclusive.
* ***vocabulary*** will contain between 1 and 50 elements, inclusive.
* Each element vocabulary will contain between 1 and 50 characters, inclusive.
* Each character of ***badPrefix***, ***badSuffix***, and ***badSubstring*** will be between 'a' and 'z', inclusive.
* Each character in ***vocabulary*** will be between 'a' and 'z', inclusive.
* All elements of vocabulary will be distinct.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

bug bug bug

buggy debugger debug

a b c

a b tco

cut sore scar

scary oscar

bar else foo

foofoofoo foobar elsewhere

pre s all

all coders be prepared for the challenge phase

**Output Example**

3

3

0

1

3

**Calc Test**

You are developing a new software calculator. During the testing phase of the software you have found that the test cases use different symbols as the decimal point of floating numbers. Moreover some test cases contain useless space symbols. Now you want to bring the numbers to a unified format.

You will be given a sequence of numbers. Remove all space symbols (ASCII code 32) from the given numbers and replace each non-digit symbol with a dot symbol ('.'). You should not make any other changes to the numbers.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with a integer N indicating the number of lines of numbers. Then follow N lines, each line with a string “number” of que given sequence ***numbers***. You can assume the following conditions will always be true

* ***numbers*** will have between 1 and 50 elements, inclusive.
* Each element of ***numbers*** will contain between 1 and 50 characters, inclusive.
* Each character in ***numbers*** will have ASCII code between 32 and 127, inclusive.
* Each element of ***numbers*** will contain at most one non-space non-digit symbol.
* Each element of ***numbers*** will contain at least one digit.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

3

1.5

2$ 3

12 3

6

,5

3,

.5

3.

000,000

000 000

3

263C45233

2364A56

B273664

**Output Example**

1.5

2.3

123

.5

3.

.5

3.

000.000

000000

263.45233

2364.56

.273664

**Dancing Sentence**

A sentence is called dancing if its first letter is uppercase and the case of each subsequent letter is the opposite of the previous letter. Spaces should be ignored when determining the case of a letter. For example, "A b Cd" is a dancing sentence because the first letter ('A') is uppercase, the next letter ('b') is lowercase, the next letter ('C') is uppercase, and the next letter ('d') is lowercase.

You will be given a sentence. Turn the sentence into a dancing sentence by changing the cases of the letters where necessary. All spaces in the original sentence must be preserved.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with the given sentence. You can assume the following conditions will always be true:

* The given sentence will contain between 1 and 50 characters, inclusive.
* Each character in the given sentence will be a letter ('A'-'Z','a'-'z') or a space (' ').
* The given sentence will contain at least one letter ('A'-'Z','a'-'z').

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

This is a dancing sentence

This is a dancing sentence

aaaaaaaaaaa

z

**Output Example**

ThIs Is A dAnCiNg SeNtEnCe

ThIs Is A dAnCiNg SeNtEnCe

AaAaAaAaAaA

Z

**Easy Conversion Machine**

We have a String, let’s call it ***originalWord***. Each character of ***originalWord*** is either 'a' or 'b'. Timmy claims that he can convert it to finalWord using exactly ***k*** moves. In each move, he can either change a single 'a' to a 'b', or change a single 'b' to an 'a'.

You are given the Strings ***originalWord*** and ***finalWord***, and the integer value ***k***. Determine whether Timmy may be telling the truth. If there is a possible sequence of exactly ***k*** moves that will turn ***originalWord*** into ***finalWord***, print "POSSIBLE" (quotes for clarity). Otherwise, print "IMPOSSIBLE".

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with ***originalWord***, ***finalWord*** and ***k***, each token will be separated by a single space and there will not be leading or trailing spaces. You can assume the following conditions will always be true:

* Timmy may change the same letter multiple times. Each time counts as a different move.
* ***originalWord*** will contain between 1 and 50 characters, inclusive.
* ***finalWord*** and ***originalWord*** will contain the same number of characters.
* Each character in ***originalWord*** and ***finalWord*** will be 'a' or 'b'.
* ***k*** will be between 1 and 100, inclusive.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

aababba bbbbbbb 2

aabb aabb 1

aaaaabaa bbbbbabb 8

aaa bab 4

aababbabaa abbbbaabab 9

**Output Example**

IMPOSSIBLE

IMPOSSIBLE

POSSIBLE

POSSIBLE

IMPOSSIBLE

**Falling Factorial Power**

A number ***n*** taken to the falling factorial power ***k*** is defined as n\*(n-1)\*...\*(n-k+1). We will denote it by n^^k. For example, 7^^3=7\*6\*5=210. By definition, n^^1=n.

We will now continue this definition to the non-positive values of ***k*** using the following fact: (n-k)\*(n^^k)=n^^(k+1), or, in other words, n^^k=(n^^(k+1))/(n-k). It is directly derived from the above definition.

By using it, we find:

n^^0=n^^1/(n-0)=1,

n^^(-1)=n^^0/(n+1)=1/(n+1),

n^^(-2)=1/(n+1)/(n+2),

and, in general, n^^(-k)=1/(n+1)/(n+2)/.../(n+k).

For example, 3^^(-1)=1/4=0.25, 2^^(-3)=1/3/4/5=1/60=0.016666...

Given a positive integer ***n*** (1<=n<=10) and an integer ***k*** (-5<=n<=5), find and print a real number containing the value of ***n*** taken to the falling factorial power of ***k***.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with ***n*** and ***k***.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example. The answer must be rounded and printed with sex decimal positions.

**Input Example**

7 3

10 1

5 0

3 -1

2 -3

**Output Example**

210.000000

10.000000

1.000000

0.250000

0.016667

**General Chess**

You have decided that too many people do not know how to play chess. So, in an effort to teach the rules you must write some software that helps to understand how chess-pieces affect one another. Your current project involves the knight and its ability to threaten one or more pieces at once. The knight has an unusual style of "jumping" around the board. One move consists of traveling two squares in one of the four cardinal directions, followed by one square perpendicular to the original direction. For example, if a knight is on (0,0), it may move to (2,1), (2,-1), (1,2), (1,-2), (-2, 1), (-2,-1), (-1,2), or (-1,-2). In addition, if a piece is on any of those locations, it is threatened by the knight on (0,0).

You will be given a sequence of pieces; ***pieces***, where each element is a comma delimited set of coordinates. Every element in pieces is formatted as "<x-coordinate>,<y-coordinate>" (quotes and angle brackets for clarity). Calculate and print a sequence of lines where each line represents a position from which a knight threatens every piece in ***pieces***. Your printed sequence of lines must be in the same format as pieces and sorted in increasing order by the x-coordinate. If two sets of coordinates have the same x-coordinate, the one with the smaller y-coordinate must come first.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line an Integer N, indicating the numbers of lines that follow, each of the next N lines will be an element of ***pieces.*** You can assume the following conditions will always be true:

* **pieces** will contain between 1 and 8 elements, inclusive.
* Each element in **pieces** will be formatted as "<x-coordinate>,<y-coordinate>" (quotes and angle brackets for clarity).
* Each <x-coordinate> will be an integer between -10000 and 10000, inclusive and will not contain leading zeros.
* Each <y-coordinate> will be an integer between -10000 and 10000, inclusive and will not contain leading zeros.
* Each element in **pieces** will be unique.

**Output Specification**

For each test case, print several lines with the answer; follow the format on the output example.

**Input Example**

1

0,0

2

2,1

-1,-2

2

0,0

2,1

3

-1000,1000

-999,999

-999,997

**Output Example**

8 found

-2,-1

-2,1

-1,-2

-1,2

1,-2

1,2

2,-1

2,1

2 found

0,0

1,-1

0 found

1 found

-1001,998

8 found

-10002,-10001

-10002,-9999

-10001,-10002

-10001,-9998

-9999,-10002

-9999,-9998

-9998,-10001

-9998,-9999

**Haar1D**

The Haar wavelet transform is possibly the earliest wavelet transform, introduced by Haar in 1909. The 1-dimensional version of this transform operates on a sequence of integer data as follows: First separate the sequence into pairs of adjacent values, starting with the first and second values, then the third and fourth values, etc. Next, calculate the sums of each of these pairs, and place the sums in order into a new sequence. Then, calculate the differences of each of the pairs (subtract the second value of each pair from the first value), and append the differences in order to the end of the new sequence. The resulting sequence is a level-1 transform. Note that this requires the input sequence to have an even number of elements.

The above describes a level-1 transform. To perform a level-2 transform, we repeat the above procedure on the first half of the sequence produced by the level-1 transform. The second half of the sequence remains unchanged from the previous level. This pattern continues for higher level transforms (i.e., a level-3 transform operates with the first quarter of the sequence, and so on). Note that this is always possible when the number of elements is a power of 2.

Given a sequence of integers; ***data*** and an integer ***L***. Find and print the level-L Haar transform of the data.

**Input Specification**

The input will consist in several test cases; each test case will consist of two lines, a line with the ***data***, and a line with ***L***. You can assume the following conditions will always be true

* The given ***data*** will contain exactly 2, 4, 8, 16 or 32 elements.
* Each element of data will be between 0 and 100 inclusive.
* ***L*** will be between 1 and log2(# of elements in data) inclusive.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

1 2 3 5

1

1 2 3 5

2

1 2 3 4 4 3 2 1

3

94 47 46 28 39 89 75 4 28 62 69 89 34 55 81 24

2

**Output Example**

3 8 -1 -2

11 -5 -1 -2

20 0 -4 4 -1 -1 1 1

215 207 248 194 67 49 -68 -16 47 18 -50 71 -34 -20 -21 57

**Important Sequence**

Little Rudolph had an important sequence of positive integers. The sequence consisted of N positive integers a0, a1, .., aN-1.

Rudolph wrote the sequence onto the blackboard in the classroom. While Rudolph had gone out, little Arthur came into the classroom and saw the sequence. Arthur likes to play with numbers as much as he likes to give his friends puzzles. So he did the following:

First, he wrote a '+' or a '-' between each pair of consecutive numbers (possibly using different signs for different pairs of numbers).

Next, for each sign he computed the result of the corresponding operation and wrote it under the sign. I.e., if he used the '+' sign between ai and ai+1, he would write the sum ai+ai+1 under this '+' sign. Similarly, if he used the '-' sign between ai and ai+1, he would write the difference ai-ai+1. In this way he obtained a new sequence of N-1 numbers b0, b1, .., bN-2.

Finally, he erased the original sequence. Now there was only the operator sequence o0, o1, .., oN-2 and the resulting number sequence b0, b2, .., bN-2 left on the blackboard.

For example, if the original sequence was {1, 2, 3, 4}, and Arthur wrote operators {+, -, +}, then the content of the blackboard changed like this:

1 2 3 4 -> 1 + 2 - 3 + 4 -> 1 + 2 - 3 + 4 -> + - +

3 -1 7 3 -1 7

When Rudolph returned, he was shocked as his important sequence had disappeared. Arthur quickly told him what operations he had performed and that Rudolph has to simply reconstruct the orginal sequence.

Unfortunately, little Arthur did not realize that it is not necessarily possible to determine the original sequence uniquely. For example, both original sequences {1, 2, 3, 4} and {2, 1, 2, 5} lead to the same sequence {3, -1, 7} when operator sequence is {+, -, +}.

The only thing Rudolph remembers about his original sequence is that all the integers were positive. Rudolph now wants to count all sequences of positive integers that match the blackboard. You are given an integer sequence called ***B*** and line of operators called ***operators*** both containing N-1 elements. The i-th element of ***B*** is the number bi and i-th element of operators will be '+' or '-', meaning that the i-th operator is + or -, respectively. Find and print the number of different positive integer sequences A that lead to sequence ***B*** when operators ***operators*** are used in the way described. If there are infinitely many such sequences, just print -1. Note that there may be test cases where no valid sequence A exists. For such test cases the correct value to print is 0.

Input Specification

The input will consist in several test cases; each test case will consist of two lines, a line with the sequence B, and a line with the ***operators***. You can assume the following conditions will always be true:

* It is guaranteed that the correct answer will always fit into the 32-bit signed integer type.
* The integer 0 (zero) is not positive. It may not occur in Rudolph's original sequence.
* ***B*** will contain between 1 and 50 elements, inclusive.
* ***operators*** will contain the same number of characters as the number of elements in ***B***.
* Each element of ***B*** will be between -1000000000 (-109) and 1000000000 (109), inclusive.
* Each character in operators will be either '+' or '-' (quotes for clarity).

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

3 -1 7

+-+

1

-

1

+

10

+

540 2012 540 2012 540 2012 540

-+-+-+-

**Output Example**

2

-1

0

9

1471

**Jaguars**

There are N animals numbered 0 to N-1 in a zoo. Each animal is a jaguar or a cat. Their heights are pairwise distinct.

Fox Jiro can't distinguish between jaguars and cats, so he asked the following question to each animal: "How many animals of the same kind as you are taller than you?" Each jaguar tells the number of jaguars taller than him, and each cat tells the number of cats taller than her. The differences of heights are slight, so Fox Jiro can't tell which animals are taller than other animals. However, each animal is able to determine which animals are taller that him and which ones are shorter.

The answer given by the i-th animal is ***answersi***. Given these numbers, find and print the number of configurations resulting in exactly those numbers, assuming everyone tells the truth. Two configurations are different if there exists an i such that the i-th animal is a jaguar in one configuration and cat in the other configuration.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with the sequence ***answer***. You can assume the following conditions will always be true:

* ***answers*** will contain between 1 and 40 elements, inclusive.
* Each element of ***answers*** will be between 0 and 40, inclusive.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

0 1 2 3 4

5 8

0 0 0 0 0 0

1 0 2 0 1

1 0 1

**Output Example**

2

0

0

8

0

**Kingdom And Trees**

King Dengklek once planted N trees, conveniently numbered 0 through N-1, along the main highway in the Kingdom of Ducks. As time passed, the trees grew beautifully. Now, the height of the i-th tree is ***heights[i]*** units.

King Dengklek now thinks that the highway would be even more beautiful if the tree heights were in strictly ascending order. More specifically, in the desired configuration the height of tree i must be strictly smaller than the height of tree i+1, for all possible i. To accomplish this, King Dengklek will cast his magic spell. If he casts magic spell of level X, he can increase or decrease the height of each tree by at most X units. He cannot decrease the height of a tree into below 1 unit. Also, the new height of each tree in units must again be an integer.

Of course, a magic spell of a high level consumes a lot of energy. Find and print the smallest possible non-negative integer X such that King Dengklek can achieve his goal by casting his magic spell of level X.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with the given sequence of ***height***s. You can assume the following conditions will always be true:

* heights will contain between 2 and 50 elements, inclusive.
* Each elements of heights will be between 1 and 1,000,000,000, inclusive.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

9 5 11

5 8

1 1 1 1 1

548 47 58 250 2012

**Output Example**

3

0

4

251

**Land And Sea**

Bob's father bought him a toy map of islands and seas. The map is a two-dimensional grid where each cell is either 'x' or '.'. A sea is defined as a maximal connected group of '.' cells, where two '.' cells are connected if they are vertically or horizontally adjacent. An island is defined as a maximal connected group of 'x' cells, where two 'x' cells are connected if they are vertically, horizontally, or diagonally adjacent. An island has a level of 0 if it contains no other islands. An island has a level of K+1 if it contains one or more islands and the highest level of a contained island is K. An island A contains island B if A and B are different and, if you start sailing from any point of island B, you won't be able to sail out of island A (you can sail only horizontally and vertically, but not diagonally).

For example, the given map below has 5 islands with level 0 (islands 0 - 4 on the right picture) and one island with level 1 (island 5). Please note that starting at island 3, you can not sail outside island 5 (you can not sail diagonally), but its possible get out of island 1 when starting at island 4.

xxx.x...xxxxx 000.0...11111

xxxx....x...x 0000....1...1

........x.x.x ........1.4.1

..xxxxx.x...x ..55555.1...1

..x...x.xxx.x ..5...5.111.1

..x.x.x...x.. ..5.3.5...1..

..x...x...xxx ..5...5...111

...xxxxxx.... ...555555....

x............ 2............

Given seaMap, find and print several integers (may be one), where the k-th element is the number of islands of level k. The output to the problem must contain exactly (m + 1) elements, where m is the highest level of an island in the map.

**Input Specification**

The input will consist in several test cases; each test case will consist of a line with ***R*** and ***C***,**1<=R,C<=50**, the numbers of rows and cols of the given **seaMap**, the next R lines will contain the given **seaMap**. You can assume the following conditions will always be true:

* **seaMap** will contain between 1 and 50 elements, inclusive.
* Each element of **seaMap** will contain between 1 and 50 characters, inclusive.
* Each element of **seaMap** will contain the same number of characters.
* Each element of **seaMap** will contain only '.' and lowercase 'x' characters.

**Output Specification**

For each test case, print one line with the answer, follow the format on the output example.

**Input Example**

1 1

x

5 5

xxxxx

x...x

x.x.x

x...x

xxxxx

10 5

xxxxx

x...x

x.x.x

x...x

xxxxx

xxxxx

x...x

x.x.x

x...x

xxxxx

2 2

..

..

6 12

............

.......xxxx.

..xxx.x...x.

..x..x..x.x.

..x.x.x...x.

..xx...xxx..

**Output Example**

{1}

{1, 1}

{2, 1}

{}