Problem A
Advanced edit distance

Source file name: `aed.c, aed.cpp` *or* `aed.java`

The edit distance of two strings S and T is the minimum number of edit operations that need to be done to transform S into T. The valid edit operations are:

- Insert a single character at any position.
- Modify an existing character.
- Remove an existing character.

For example, the edit distance of “pantera” and “aorta” is 5, because the following chain of edits is valid (and there is no shorter chain):

“pantera” → “antera” → “aotera” → “aoera” → “aora” → “aorta”.

We define the advanced edit distance in a similar way, but adding the swap of two adjacent characters as an extra valid operation. With this setting, the advanced edit distance of “pantera” and “aorta” is 4:

“pantera” → “antera” → “antra” → “aotra” → “aorta”.

You need to write a program that calculates the advanced edit distance of two given words.

Input

The input contains several test cases. Each test case is described in a single line that contains two non-empty words, each of them of at most 1000 lowercase letters, separated by a single space. The last line of the input contains two asterisks separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with an integer representing the advanced edit distance of the two input words.

The output must be written to standard output.

Sample input	**Output for sample input**

`pantera aorta`
`zero zero`
`* *`
	`4`
	`0`

Problem B
Back to the polygon

Source file name: `backtopol.c, backtopol.cpp` *or* `backtopol.java`

A *simple* polygon is a polygon that does not overlap with itself. A *diagonal* of a simple polygon is a segment within the polygon that connects two non-consecutive vertices. A *triangulation* of a simple polygon of N edges is the drawing of exactly $N - 3$ diagonals that do not touch each other anywhere, with the possible exception of their endpoints. A triangulation divides the polygon in exactly $N - 2$ triangles that do not overlap and only touch each other along their edges.

In this problem, you are given the triangulation of a simple polygon, which means, the set of triangles in which a polygon was divided. From them, you need to reconstruct the original polygon.

Input

The input contains several test cases, each one described in several lines. The first line of each test case contains an integer N ($3 \leq N \leq 500$), the number of edges of the original polygon. Each of the next $N - 2$ lines describes one triangle in the triangulation of the polygon. Each triangle is given by six integers X_1, Y_1, X_2, Y_2, X_3 and Y_3 separated by single spaces, where X_i and Y_i are the coordinates in the XY plane of the i-th vertex of the triangle ($-1000 \leq X_i, Y_i \leq 1000$). The triangles and their vertices are not given in any specific order. The last line of the input contains a single -1 and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with $2N$ integers separated by single spaces. These integers must represent the coordinates in the XY plane of the vertices of the original polygon, in clockwise order. To make the output unique, the first vertex to be listed is the one with the smallest X coordinate, and if there are many of those, the one with the smallest Y coordinate among them.

The output must be written to standard output.

Sample input	**Output for sample input**
5	0 0 0 9 5 13 10 9 10 0
0 0 10 9 10 0	0 1 1 1 1 0
10 9 0 9 0 0	-2 3 2 3 2 2 1 2 2 1 2 0 0 -1 2 -1 2 -2 -1 -2
10 9 0 9 5 13	
3	
0 1 1 1 1 0	
10	
-1 -2 2 -2 2 -1	
-1 -2 0 -1 -2 3	
-2 3 2 3 1 2	
0 -1 2 1 1 2	
-1 -2 2 -1 0 -1	
2 1 0 -1 2 0	
2 3 2 2 1 2	
0 -1 -2 3 1 2	
-1	

Problem C
Charly & Nito

Source file name: `charly.c, charly.cpp` *or* `charly.java`

Charly and Nito are friends and they like to be together at a nice bar in Palermo Hollywood. About at 3 a.m. they start to feel sleepy and want to go home. They want to get home quickly so each of them uses a path that minimizes the distance to his home. However, Charly and Nito also like to walk together while they talk about the "good old times", so they want to walk together as much as possible.

Charly and Nito live in a city that can be modeled as a set of streets and junctions. Each street connects a pair of distinct junctions and can be walked in both directions. No two streets connect the same pair of junctions. Charly and Nito do not live together, and they do not live at the bar. There is at least one path from the bar to Charly's home; the same occurs with Nito's home.

Given information about the streets and junctions in the city, the locations of the bar, Charly's home and Nito's home, you must tell Charly and Nito the maximum distance that they can walk together without forcing them to walk more than the minimum distance from the bar to their respective homes. Charly and Nito also want to know how much each of them will walk alone.

Input

The input contains several test cases, each one described in several lines. The first line of each test case contains five integers J, B, C, N and S separated by single spaces. The value J is the number of junctions in the city ($3 \leq J \leq 5000$); each junction is identified by an integer number between 1 and J. The values B, C and N are the identifiers of the junctions where the bar, Charly's home and Nito's home are located, respectively ($1 \leq B, C, N \leq J$); these three junction identifiers are different. The value S is the number of streets in the city ($2 \leq S \leq 150000$). Each of the next S lines contains the description of a street. Each street is described using three integers E_1, E_2 and L separated by single spaces, where E_1 and E_2 identify two distinct junctions that are endpoints of the street ($1 \leq E_1, E_2 \leq J$), and L is the length of the street ($1 \leq L \leq 10^4$). You may assume that each street has a different pair of endpoints, and that there exist paths from junction B to junctions C and N. The last line of the input contains the number -1 five times separated by single spaces and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with three integers T, C and N separated by single spaces, where T is the maximum distance that Charly and Nito can walk together, C is the distance that Charly walks alone, and N is the distance that Nito walks alone.

The output must be written to standard output.

Sample input	**Output for sample input**
`5 3 2 1 6`	`20 4 3`
`3 4 10`	`4 1 1`
`4 5 10`	
`5 1 3`	
`5 2 4`	
`1 3 23`	
`2 3 24`	
`8 1 7 8 8`	
`1 2 1`	
`2 4 1`	
`2 3 1`	
`4 5 1`	
`3 5 1`	
`5 6 1`	
`6 8 1`	
`6 7 1`	
`-1 -1 -1 -1 -1`	

Problem D
Doing the word wrap

Source file name: `doingww.c, doingww.cpp` *or* `doingww.java`

You are developing a visual component for a web browser. The component is known as *textarea*, and its main functionality is to show a given text using one or more lines. Every textarea has a *linewidth* W, which is the number of characters that can fit in a single line.

The text that needs to be shown is a sequence of words. The textarea must display the text using lines of W characters, without breaking any word, and placing a single space between each pair of consecutive words that are in the same line. Any number of trailing spaces may be left at the end of each line. So the behavior of the textarea is quite simple: it keeps adding words to a line until the next word does not fit; each time this occurs, a new line is started.

With the permanent growing in the amount of information that web pages must show, you have to make a smart textarea that uses as little space as possible, even when dealing with very long texts. Given a text to show and a number of lines L, you must set the linewidth W to the minimum possible value such that the text is shown using at most L lines.

Input

The input contains several test cases, each one described in exactly two lines. The first line of each test case contains two integers L and N separated by a single space, where L is the maximum number of lines the textarea can have ($1 \le L \le 10^8$), and N is the number of words the text to show is made of ($1 \le N \le 10^5$). The second line contains the text to show, formed by N non-empty words of at most 25 lowercase letters each, separated by an arbitrary number of spaces. The last line of the input contains the number -1 twice separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with an integer W representing the minimum linewidth such that the textarea has at most L lines.

The output must be written to standard output.

Sample input	**Output for sample input**

`1 2`
`hello     word`
`2 2`
`racing club`
`-1 -1`
	`10`
	`6`

Problem E
Edit distance

Source file name: `edit.c, edit.cpp` *or* `edit.java`

The edit distance of two strings S and T is the minimum number of edit operations that need to be done to transform S into T. The valid edit operations are:

- Insert a single character at any position.
- Modify an existing character.
- Remove an existing character.

For example, the edit distance of “pantera” and “aorta” is 5, because the following chain of edits is valid (and there is no shorter chain):

“pantera” → “antera” → “aotera” → “aoera” → “aora” → “aorta”.

In this problem, given a value K and a word S, we need to construct a word T such that the edit distance of S and T is at most K. There are of course several possibilities for that, so we will ask that you choose the word T that comes first alphabetically.

A word always comes alphabetically after any proper prefix. Among two words that are not prefixes of each other, the one that comes first alphabetically is the one that has, in the first position at wich they differ from left to right, a letter closest to the beginning of the alphabet. Notice that the empty word (that has zero characters) is a valid word and is alphabetically before any other word.

Input

The input contains several test cases. Each test case is described in a single line that contains an integer K ($0 \leq K \leq 1000$) and a non-empty word S of at most 1000 lowercase letters, separated by a single space. The last line of the input contains the number -1 and an asterisk separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with a word T of lowercase letters such that the edit distance of S and T is at most K. If there are several words in that situation, output the one that comes first alphabetically.

The output must be written to standard output.

Sample input	**Output for sample input**
`4 abcadef`	`aaaaaadef`
`1000 zero`	
`0 zero`	`zero`
`-1 *`	

Problem F
Feanor the elf

Source file name: `feanor.c, feanor.cpp` *or* `feanor.java`

Feanor is an elf, and of course, he really likes arrows and bows. Surprisingly enough, Feanor has a laptop, but he knows nothing about programming, so he requires your help.

Feanor lives in a tower of height H, and he loves throwing arrows from the top of it. He had a good amount of intesive training and he knows that he always throws his arrows with the same initial speed V. He wants you to make a program that given H and V returns the maximum distance that a Feanor's arrow can reach when it hits the ground, measured from the base of the tower. With this information, he will be able to place a nice circular fence to prevent deoriented little elves from being killed.

Newtonian laws apply in Feanor's world and the gravity has the same strength as in ours. These laws can be summarized as follows:

- The position of Feanor is assumed to be a point. The same occurs with the position of his arrow at each moment in time.
- The initial speed V of the arrow can be expressed as $V_x^2 + V_y^2 = V^2$, where V_x and V_y are the horizontal and vertical components of V, respectively. Speed V_x is always non-negative, while speed V_y is positive if the arrow is thrown up, and negative if the arrow is thrown down.
- The initial position of the arrow is the position of Feanor.
- The horizontal position of the arrow (relative to Feanor's position) at time t is $x(t) = V_x t$.
- The vertical position of the arrow (relative to Feanor's position) at time t is $y(t) = V_y t - gt^2/2$, where $g = 9.8$ m/s^2.

Input

The input contains several test cases. Each test case is described in a single line that contains two integers V and H separated by a single space. The value V is the initial speed of Feanor's arrow measured in m/s ($0 \leq V \leq 1000$), while the value H is the tower's height in meters ($0 \leq H \leq 1000$). The last line of the input contains the number -1 twice separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with the radius of Feanor's fence in meters, rounded up to 6 decimal digits (he wants to be sure that he doesn't kill those cute little elves).

The output must be written to standard output.

Sample input	**Output for sample input**
`1 0`	`0.102041`
`10 0`	`10.204082`
`100 0`	`1020.408163`
`1000 0`	`102040.816327`
`-1 -1`	

Problem G
G key

Source file name: `gkey.c, gkey.cpp` *or* `gkey.java`

Leandro and Fede are traveling by train and to spend some time they decided to start playing the guitar. They want to play together some songs, but Fede's memory is not working well because he caught a little cold. To work it out, Leandro wants to show to Fede some music scores of several punk rock songs. A punk rock song is a sequence of notes, and there are twelve possible notes, divided in two groups. The first group has seven natural notes called A, B, C, D, E, F and G, while the second group has five alterations called A#, C#, D#, F# and G#.

The way to draw a music score is as follows: you start with an empty one, and then you draw note by note from left to right in the same order they appear in the song. Each line is divided in four groups, each group having room for four notes.

As punk rock is quite simple the drawing is even easier. Natural notes have always the same shape called *eighth note*, and they differ only in the height at which they are drawn. Alterations are drawn like natural notes but with a preceeding "#" sign. To make it simpler, Leandro will avoid the drawing of the G key.

Can you help Leandro writing a program for drawing punk rock songs given the sequence of notes?

Input

The input contains several test cases. Each test case is described in a single line that contains the number of notes N ($1 \leq N \leq 100$), followed by the sequence of N notes. Each note is one of A, A#, B, C, C#, D, D#, E, F, F#, G and G#. Values in each line are separated by single spaces. The last line of the input contains a single -1 and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output the music score of the input song and print a blank line after each test case (even after the last one). You have to follow the sample input and output for drawing the music scores. Every score line has the same background formed by the characters "`|`" (pipe) and "`-`" (hyphen). They differ in the borders (first and last borders are doubled), and of course in the notes they have inside. Each eighth note is drawn consecutively as in the sample, and the different heights are those shown. Alterations are preceded by a character "`#`" (sharp sign). There must be no trailing spaces at the end of printed lines, neither empty score lines (without notes inside).

The output must be written to standard output.

Sample input

```
36 E F F# G G# A A# B C C# D D# E F F# G G# A A# B C C# D D# E F F# G G# A A# B C C# D D#
2 B# B#
-1
```

Output for sample input

```
||                        |                        |                        |                        |
||------------------------|------------------------|--------------|\----|\--|------------------------|
||                        |                        |  |\    |\    |     |   |                        |
||------------------------|--------------------|\--|--|-----|----x|---#x|---|------------------------|
||                        |        |\    |\    |   | x|   #x|               |                        |
||--------------------|\--|--|\----|-----|----x|---|------------------------|--------------------|\--|
||        |\    |\    |   |  |    x|   #x|         |                        |        |\    |\    |   |
||--|\----|-----|----x|---|#x|---------------------|------------------------|--|\----|-----|----x|---|
||  |    x|   #x|         |                        |                        |  |    x|   #x|         |
||-x|---------------------|------------------------|------------------------|-x|---------------------|
||                        |                        |                        |                        |
 |                        |                        |                        |                        |
 |------------------------|--------------|\----|\--|------------------------|------------------------|
 |                        |  |\    |\    |     |   |                        |                        |
 |--------------------|\--|--|-----|----x|---#x|---|------------------------|--------------------|\--|
 |        |\    |\    |   | x|   #x|               |                        |        |\    |\    |   |
 |--|\----|-----|----x|---|------------------------|--------------------|\--|--|\----|-----|----x|---|
 |  |    x|   #x|         |                        |        |\    |\    |   |  |    x|   #x|         |
 |#x|---------------------|------------------------|--|\----|-----|----x|---|#x|---------------------|
 |                        |                        |  |    x|   #x|         |                        |
 |------------------------|------------------------|-x|---------------------|------------------------|
 |                        |                        |                        |                        |
 |                        |                        |                        |                        ||
 |--------------|\----|\--|------------------------|------------------------|------------------------||
 |  |\    |\    |     |   |                        |                        |                        ||
 |--|-----|----x|---#x|---|------------------------|------------------------|------------------------||
 | x|   #x|               |                        |                        |                        ||
 |------------------------|------------------------|------------------------|------------------------||
 |                        |                        |                        |                        ||
 |------------------------|------------------------|------------------------|------------------------||
 |                        |                        |                        |                        ||
 |------------------------|------------------------|------------------------|------------------------||
 |                        |                        |                        |                        ||

||                        |                        |                        |                        ||
||------------------------|------------------------|------------------------|------------------------||
||                        |                        |                        |                        ||
||--|\----|\--------------|------------------------|------------------------|------------------------||
||  |     |               |                        |                        |                        ||
||#x|---#x|---------------|------------------------|------------------------|------------------------||
||                        |                        |                        |                        ||
||------------------------|------------------------|------------------------|------------------------||
||                        |                        |                        |                        ||
||------------------------|------------------------|------------------------|------------------------||
||                        |                        |                        |                        ||
```

Problem H
Heptadecimal numbers

Source file name: `hepta.c, hepta.cpp` *or* `hepta.java`

The Factory of Computer Enhaced Numbers (FCEN) has asked its Development Comitee (DC) to come up with a way to handle numbers written in base 17.

As everybody knows, base 17 is very important for many scientific applications, as well as for engineering and other practical uses. Numbers in base 17 can be tough, but are kind and soft if treated appropiately.

Numbers in base 17 are written by using a set of 17 characters: digits 0 to 9 with their usual values, and uppercase letters A to G that have values from 10 to 16, respectively.

Base 17, probably because its basement on a prime number, does not require numbers to start with a non-zero digit, so each number has many representations. For instance, the decimal number 117 can be written as $6F$, but also as $06F$ or even $00000006F$.

Because of this leading-zeroes thing, heptadecimal numbers are hard to compare. As a member of the FCEN-DC, you were asked to write a program that helps in this difficult and challenging task.

Input

The input contains several test cases. Each test case is described in a single line that contains two non-empty strings of at most 10^5 heptadecimal digits, separated by a single space. The last line of the input contains two asterisks separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with the sign "<" if the first heptadecimal number is smaller than the second one, the sign ">" if the first heptadecimal number is greater than the second one, or the sign "=" if both heptadecimal numbers are equal.

The output must be written to standard output.

Sample input	**Output for sample input**

`006F B3B`
`0000 0`
`* *`
	`<`
	`=`

Problem I
Indicator of progression

Source file name: `indicator.c, indicator.cpp` *or* `indicator.java`

When dealing with a long task, computers often provide a progress indicator to help users estimate how much longer they will have to wait. This is especially useful when copying a large number of data files from one drive to another.

In the Institute of Computer Power Control (ICPC) are very concerned about their brand new file copier, which they think will change forever the way people copy files. While this is a great accomplishment for the engineers in ICPC, the lack of a progress indicator is threatening the future of the project and the well being of most computer users around the world!

The Supremum Principal Director Manager of ICPC has called you personally to ensure you are up for the task. The interface provided by the developing team of the file copier only gives two integers M and N. M is the number of files that have already been copied, and N is the total number of files to be copied. Using this information, you must write a module that displays the progress indicator.

The indicator must be drawn as a string of exactly 20 characters. The first K of them must be asterisks ("`*`") and the rest must be hyphens ("`-`"). The number K must be chosen in such a way that $K/20$ correctly approximates M/N; this means that the distance between the two mentioned fractions is minimum. If there is more than one possible value for K, the greatest one must be chosen.

Also, for more precision, a number P without leading zeroes and followed by a percentage sign ("`%`") must be written on top of the described indicator. Since the goal is to represent the finished percentage, the number P must be such that $P/100$ correctly approximates M/N, with the same policy as before. The finished percentage must be centered on top of the display. This means that if possible, the same number of display characters ("`*`" or "`-`") must be seen to the left and to the right of the percentage; if this is not possible, exactly one extra character must be seen to the left.

Input

The input contains several test cases. Each test case is described in a single line that contains two integers M and N as explained above ($0 \leq M \leq N \leq 10^9$ and $N \neq 0$). These values are separated by a single space. The last line of the input contains the number -1 twice separated by a single space and should not be processed as a test case.

The input must be read from standard input.

Output

For each test case output a single line with exactly 20 characters representing the mentioned display.

The output must be written to standard output.

Sample input	**Output for sample input**
2 5	********-40%--------
2 6	*******--33%--------
0 10	---------0%---------
-1 -1	