ANZAC 2013

Round 2

(not the

but kind of similar)

Compiled by Tim French and James Hales from various sources.

Acknowledgements

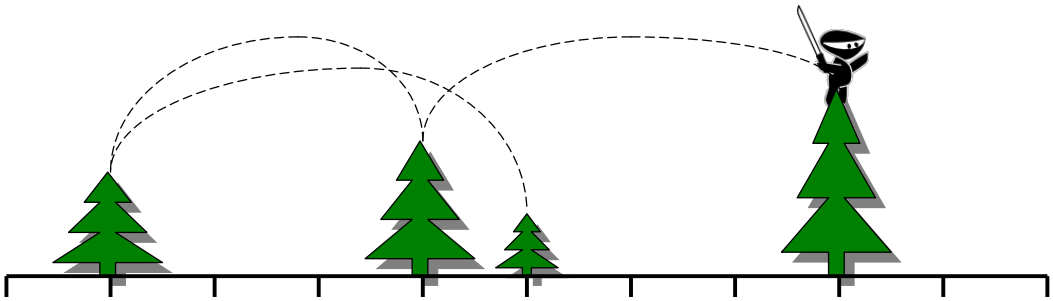
All problems were sourced from the internet and all credit should rest with the original authors. The problems in this set came from the following sources:

A: *The Ninja Way* came from the 2009 South East USA Programming Competition (Problem F)
B: *Scramble Sort* came from the 2000 Greater New York Regional Programming Competition (Problem D)
C: *The Robot Way* came from the 2009 South East USA Programming Competition (Problem H)
D: *Bit Counting* came from the 2010 South East USA Programming Competition (Problem B)
E: *Digital Roots* came from the 2000 Greater New York Regional Programming Competition (Problem C)
F: *Maximum Square* came from the 2010 South East USA Programming Competition (Problem E)
G: *Joint Venture* came from the North West European Regional Programming Competition (Problem J)
H: *Palindrometer* came from the 2010 South East USA Programming Competition (Problem F)
I: *Key Insight* came from the North West European Regional Programming Competition (Problem K)
J: *Guards!* came from the North West European Regional Programming Competition (Problem G)

Problem A: The Ninja Way

As we all know, Ninjas travel by jumping from treetop to treetop. A clan of Ninjas plans to use **N** trees to hone their tree hopping skills. They will start at the shortest tree and make **N-1** jumps, with each jump taking them to a taller tree than the one they're jumping from. When finished, they will have been on every tree exactly once, traversing them in increasing order of height, and ending up on the tallest tree.

The ninjas can travel for at most a certain horizontal distance **D** in a single jump. To make this as much fun as possible, the Ninjas want to maximize the distance between the positions of the shortest tree and the tallest tree.



The ninjas are going to plant the trees subject to the following constraints.

- All trees are to be planted along a one-dimensional path.
- Trees must be planted at integer locations along the path, with no two trees at the same location.
- Trees must be arranged so their planted ordering from left to right is the same as their ordering in the input. They must NOT be sorted by height, or reordered in any way. They must be kept in their stated order.
- The Ninjas can only jump so far, so every tree must be planted close enough to the next taller tree. Specifically, they must be no further than **D** apart on the ground (the difference in their heights doesn't matter).

Given **N** trees, in a specified order, each with a distinct integer height, help the ninjas figure out the maximum possible distance they can put between the shortest tree and the tallest tree, and be able to use the trees for training.

Input

There will be multiple test cases. Each test case begins with a line containing two integers **N** ($1 \le N \le 1000$) and **D** ($1 \le D \le 10^6$). The next **N** lines each contain a single integer, giving the heights of the **N** trees, in the order that they should be planted. Within a test case, all heights will be unique. The last test case is followed by a line with two **0**'s.

Output

For each test case, output a line with a single integer representing the maximum distance between the shortest and tallest tree, subject to the constraints above, or `-1` if it is impossible to lay out the trees. Do not print any blank lines between answers.

Sample Input

```
4 4
20
30
10
40
5 6
20
34
54
10
15
4 2
10
20
16
13
0 0
```

Sample Output

```
3
3
-1
```

Problem B: Scramble Sort

Background

In this problem you will be given a series of lists containing both words and numbers. The goal is to sort these lists in such a way that all words are in alphabetical order and all numbers are in numerical order. Furthermore, if the n^{th} element in the list is a number it must remain a number, and if it is a word it must remain a word.

Input

The input will contain multiple lists, one per line. Each element of the list will be separated by a comma followed a space, and the list will be terminated by a period. The input will be terminated by a line containing only a single period.

Output

For each list in the input, output the scramble sorted list, separating each element of the list with a comma followed by a space, and ending the list with a period.

Example

Input

```
0.
banana, strawberry, OrAnGe.
Banana, StRaWbErRy, orange.
10, 8, 6, 4, 2, 0.
x, 30, -20, z, 1000, 1, Y.
50, 7, kitten, puppy, 2, orangutan, 52, -100, bird, worm, 7, beetle.
.
```

Output

```
0.
banana, OrAnGe, strawberry.
Banana, orange, StRaWbErRy.
0, 2, 4, 6, 8, 10.
x, -20, 1, Y, 30, 1000, z.
-100, 2, beetle, bird, 7, kitten, 7, 50, orangutan, puppy, 52, worm.
```

This page is deliberately blank.

Problem C: The Robot Way

You have entered a robot in a Robot Challenge. A course is set up in a 100m by 100m space. Certain points are identified within the space as targets. They are ordered – there is a target 1, a target 2, etc. Your robot must start at (0,0). From there, it should go to target 1, stop for 1 second, go to target 2, stop for 1 second, and so on. It must finally end up at, and stop for a second on, (100,100).

Each target except (0,0) and (100,100) has a time penalty for missing it. So, if your robot went straight from target 1 to target 3, skipping target 2, it would incur target 2's penalty. Note that once it hits target 3, it cannot go back to target 2. It must hit the targets in order. Since your robot must stop for 1 second on each target point, it is not in danger of hitting a target accidentally too soon. For example, if target point 3 lies directly between target points 1 and 2, your robot can go straight from 1 to 2, right over 3, without stopping. Since it didn't stop, the judges will not mistakenly think that it hit target 3 too soon, so they won't assess target 2's penalty. Your final score is the amount of time (in seconds) your robot takes to reach (100,100), completing the course, plus all penalties. Smaller scores are better.

Your robot is very maneuverable, but a bit slow. It moves at 1 m/s, but can turn very quickly. During the 1 second it stops on a target point, it can easily turn to face the next target point. Thus, it can always move in a straight line between target points.

Because your robot is a bit slow, it might be advantageous to skip some targets, and incur their penalty, rather than actually maneuvering to them. Given a description of a course, determine your robot's best (lowest) possible score.

The Input

There will be several test cases. Each test case will begin with a line with one integer, **N** ($1 \le N \le 1000$) which is the number of targets on the course. Each of the next **N** lines will describe a target with three integers, **X**, **Y** and **P**, where (**X,Y**) is a location on the course ($1 \le X,Y \le 99$, **X** and **Y** in meters) and **P** is the penalty incurred if the robot misses that target ($1 \le P \le 100$). The targets will be given in order – the first line after **N** is target 1, the next is target 2, and so on. All the targets on a given course will be unique – there will be at most one target point at any location on the course. End of input will be marked by a line with a single **0**.

The Output

For each test case, output a single decimal number, indicating the smallest possible score for that course. Output this number rounded (NOT truncated) to three decimal places. Print each answer on its own line, and do not print any blank lines between answers.

Sample Input

```
1
50 50 20
3
30 30 90
60 60 80
10 90 100
3
30 30 90
60 60 80
10 90 10
0
```

Sample Output

```
143.421
237.716
154.421
```

Problem D: Bit Counting

Start with an integer, N_0, which is greater than 0. Let N_1 be the number of ones in the binary representation of N_0. So, if N_0=27, N_1=4. For all i>0, let N_i be the number of ones in the binary representation of N_{i-1}. This sequence will always converge to one. For any starting number, N_0, let K be the minimum value of i>=0 for which N_i=1. For example, if N_0=31, then N_1=5, N_2=2, N_3=1, so K=3.

Given a range of consecutive numbers, and a value X, how many numbers in the range have a K value equal to X?

Input

There will be several test cases in the input. Each test case will consist of three integers on a single line:

LO HI X

Where **LO** and **HI** ($1 \le$ **LO** $\le$ **HI** $\le 10^{18}$) are the lower and upper limits of a range of integers, and **X** ($0 \le$ **X** ≤ 10) is the target value for K. The input will end with a line with three 0s.

Output

For each test case, output a single integer, representing the number of integers in the range from **LO** to **HI** (inclusive) which have a K value equal to **X** in the input. Print each integer on its own line with no spaces. Do not print any blank lines between answers.

Sample Input

```
31 31 3
31 31 1
27 31 1
27 31 2
1023 1025 1
1023 1025 2
0 0 0
```

Sample Output

```
1
0
0
3
1
1
```

This page is deliberately blank.

Problem E: Digital Roots

Background

The *digital root* of a positive integer is found by summing the digits of the integer. If the resulting value is a single digit then that digit is the digital root. If the resulting value contains two or more digits, those digits are summed and the process is repeated. This is continued as long as necessary to obtain a single digit.

For example, consider the positive integer 24. Adding the 2 and the 4 yields a value of 6. Since 6 is a single digit, 6 is the digital root of 24. Now consider the positive integer 39. Adding the 3 and the 9 yields 12. Since 12 is not a single digit, the process must be repeated. Adding the 1 and the 2 yeilds 3, a single digit and also the digital root of 39.

Input

The input file will contain a list of positive integers, one per line. The end of the input will be indicated by an integer value of zero.

Output

For each integer in the input, output its digital root on a separate line of the output.

Example

Input	Output
24 39 0	6 3

This page is deliberately blank.

Problem F: Maximum Square

Given an NxM matrix of all 1s and 0s, find the largest submatrix which is a square containing all 1s.

Input

There will be several test cases in the input. Each test case will begin with two integers, **N** and **M** (1 ≤ **N**,**M** ≤ 1,000) indicating the number of rows and columns of the matrix. The next N lines will each contain M space-separated integers, guaranteed to be either 0 or 1. The input will end with a line with two 0s.

Output

For each test case, print a single integer, indicating the width (and height) of the largest square of all 1s, or 0 if there are no 1s. Print no extra spaces, and do not print any blank lines between answers.

Sample Input

```
4 5
0 1 0 1 1
1 1 1 1 1
0 1 1 1 0
1 1 1 1 1
3 4
1 1 1 1
1 1 1 1
1 1 1 1
6 6
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0
```

Sample Output

```
3
3
0
```

This page is deliberately blank.

Problem G: Joint Venture

Liesbeth and Jan are building a robot for a course project and have discovered that they need to fit two pieces of Lego into an opening.

The opening is x centimetres wide and the sum of the lengths of the two pieces has to be *precisely* equal to the width of the opening, or else the robot will break during the project demonstration, with catastrophic consequences for the grades of the two students.

Photo by Alan Chia

Luckily, Liesbeth and Jan were able to sneak into the physics laboratory late one night to measure the lengths of their remaining Lego pieces very accurately. Now they just need to select two pieces that will fit the opening perfectly.

Input

For each test case, you get:

- a line containing one positive integer: x, denoting the width of the opening in centimetres, with $1 \leq x \leq 20$.
- a line containing one non-negative integer: n, denoting the remaining number of Lego pieces Liesbeth and Jan have access to, with $0 \leq n \leq 1000000$.
- n lines containing positive integers ℓ, denoting lengths of Lego pieces in nanometres. Liesbeth and Jan have told you that no piece of Lego is longer than 10 centimetres, or 100000000 nanometres.

Output

For each test case, a row containing the word '`danger`' if no two pieces of Lego exist that precisely fit into the opening, or '`yes` ℓ_1 ℓ_2', with $\ell_1 \leq \ell_2$, should two such pieces of lengths ℓ_1 and ℓ_2 exist.

In case multiple solutions exist, a solution maximising the size difference $|\ell_1 - \ell_2|$ must be printed.

Example

input	output
1 4 9999998 1 2 9999999	yes 1 9999999

This page is deliberately blank.

Problem H: Palindrometer

While driving the other day, John looked down at his odometer, and it read 100000. John was pretty excited about that. But, just one mile further, the odometer read 100001, and John was REALLY excited! You see, John loves palindromes – things that read the same way forwards and backwards. So, given any odometer reading, what is the least number of miles John must drive before the odometer reading is a palindrome? For John, every odometer digit counts. If the odometer reading was 000121, he wouldn't consider that a palindrome.

The Input

There will be several test cases in the input. Each test case will consist of an odometer reading on its own line. Each odometer reading will be from 2 to 9 digits long. The odometer in question has the number of digits given in the input - so, if the input is 00456, the odometer has 5 digits. There will be no spaces in the input, and no blank lines between input sets. The input will end with a line with a single 0.

The Output

For each test case, output the minimum number of miles John must drive before the odometer reading is a palindrome. This may be 0 if the number is already a palindrome. Output each integer on its own line, with no extra spaces and no blank lines between outputs.

Sample Input

```
100000
100001
000121
00456
0
```

Sample Output

```
1
0
979
44
```

This page is deliberately blank.

Problem I: Key Insight

Alice and Bob love to send each other messages, but they don't like it when other people read their messages. Your friend Charles is very interested in what Alice and Bob send to each other, but since Alice and Bob are encrypting their messages he is unable to read them, even though he is able to intercept the encrypted messages (called "ciphertext").

Recently, Charles has not only intercepted an encrypted message, he also knows the original content of this message (which is called "plaintext"). To help him decrypt future messages between Alice and Bob, you are asked to write a program that will help him look for the encryption key.

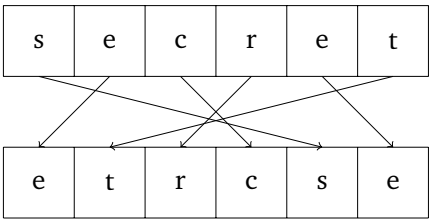


Figure 1: Example of transposition.

Charles has informed you that he knows that Alice and Bob are using a transposition block cipher. This means that for each block of k characters in the message, the characters within the block are re-ordered into one of $k!$ possible permutations during encryption. Each permutation is determined by its unique corresponding encryption key. The key corresponding to the permutation shown in Figure 1 would be some representation of $(123456) \rightarrow (514362)$. Since your only task is counting (possible) keys, the actual representation is not relevant.

Fortunately, Charles does know the block size k, and he knows that the plaintext and ciphertext that he intercepted consist of one or more full blocks of length k (i.e., no incomplete blocks) that have each been encrypted with the same key.

Given the plaintext M and ciphertext C that Charles has intercepted, your program will compute the number of possible encryption keys.

Input

For each test case, the input contains three lines:

- One line containing a positive integer k, the block size ($k \geq 1$).
- One line containing M, the plaintext ($1 \leq |M| \leq 100,\ |M|$ is a multiple of k).
- One line containing C, the ciphertext ($|C| = |M|$).

Both plaintext and ciphertext consist only of lower-case letters.

Output

For each test case, print one line containing the number of possible encryption keys of size k. This number will not exceed $2^{63} - 1$. If it is impossible to obtain M from C by a transposition cipher of block size k, print '0' (the number zero).

Example

input	output
4 treewood ertedowo 1 nwerc ncrew 6 secret etrcse 1 impossibru youdontsay	1 0 2 0

Problem J: Guards!

The royal castles in Molvania follow the design of king Sane, first of his dynasty. He ruled by divide and conquer. Therefore, all castles are built according to a hierarchical pattern based on interconnected buildings. A building consists of *halls* and *corridors* that connect halls.

Initially, a castle consists of only one building (the *main building*). When its population grows, the castle is extended as follows: A new peripheral building is constructed, attached to one of the existing buildings. Like any other building, the new building also consists of halls and corridors. An additional corridor is created to connect a hall in the existing building to a hall in the new building. That corridor is the only way to access the new building.

The number of halls in a building is at most 10.

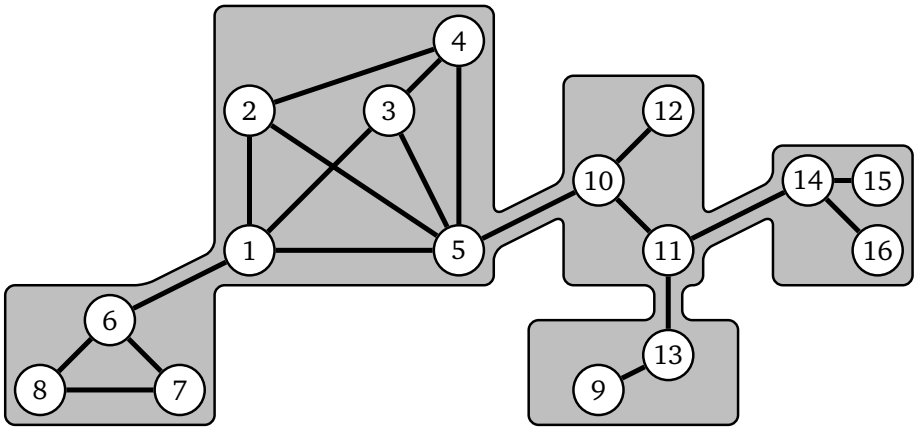


Figure 1: The castle layout of the example provided below.

In times of turmoil, the king monitors all corridors by strategically placing guards in halls. He asks you to determine the least number of guards required to monitor all corridors in the castle (as he wants to keep his personal guard as large as possible). Note that since the last fire, there are no doors in the castle, so we can safely assume that a guard placed in a hall can monitor all connecting corridors.

Input

The input contains a number of castle descriptions. Within a castle, each hall is identified by a unique number between 1 and 10000. Each castle is recursively defined, starting with a description of the main building:

1. A line containing three integers, representing the number of halls in this building ($2 \leq n \leq 10$), the number of corridors in this building ($1 \leq m \leq 45$), and the number of peripheral buildings that were later attached to this building ($0 \leq w \leq 10$).
2. For each of the m corridors:
   - A line containing two integers (each $\leq$ 10000), representing the two halls connected by this corridor. Both halls are located inside the current building.

3. For each of the w peripheral buildings:
    - A line containing two integers, describing the corridor that leads to this peripheral building. The first integer represents a hall in the current building, while the second integer represents a hall in the peripheral building.
    - The structure of the peripheral building and any newer buildings that were later attached to it, described by repeating rules 1 to 3.

The castle is fully connected: any hall is directly or indirectly reachable from any other hall. Corridors with the same start and end hall do not exist, and for every two halls there is at most one corridor between them.

Output

For each castle, print a single line containing a positive integer: the minimum number of guards to place in halls such that all corridors in the castle are monitored.

Example

input	output
5 8 2 1 2 2 4 3 4 1 3 1 5 2 5 3 5 4 5 1 6 3 3 0 6 7 7 8 8 6 5 10 3 2 2 10 11 10 12 11 13 2 1 0 13 9 11 14 3 2 0 14 15 14 16	8