

TD n°9

1 Rings and Glue (10301)

Time limit : 3.000 seconds.

Little John is in big trouble. Playing with his different-sized (and colored!) rings and glue seemed such a good idea. However, the rings now lay on the floor, glued together with some-thing that will definitely not come off with water. Surprisingly enough, it seems like no rings are actually glued to the floor, only to other rings. How about that!

You must help Little John to pick the rings off the floor before his mom comes home from work. Since the glue is dry by now, it seems like an easy enough task. This is not the case. Little John is an irrational kid of numbers, and so he has decided to pick up the largest component (most rings) of glued-together rings first. It is the number of rings in this largest component you are asked to find. Two rings are glued together if and only if they overlap at some point but no rings will ever overlap in only a single point. All rings are of the doughnut kind (with a hole in them). They can however, according to Little John, be considered “infinitely thin”.

1.1 Input

Input consists of a number (>0) of problems. Each problem starts with the number of rings, n , where $0 \leq n < 100$. After that, n rows follow, each containing a ring’s physical attributes. That is, 3 floating point numbers, with an arbitrary number of spaces between them, describing the x coordinate and y coordinate for its center and its radius. All floating point numbers fit into the standard floating point type of your favorite programming language (*e.g.*, float for C/C++). Input ends with a single row with the integer -1.

1.2 Output

Output consists of as many grammatically correct answers as there were problems, each answer, on a separate line, being ‘**The largest component contains X ring(s).**’ where X is the number of rings in the largest component.

1.3 Sample Input

```
4
0.0 0.0 1.0
-1.5 -1.5 0.5
1.5 1.5 0.5
-2.0 2.0 3.5
3
3.0 2.0 2.0
0.0 -0.5 1.0
0.0 0.0 2.0
5
-2.0 0.0 1.0
1.0 -1.0 1.0
0.0 1.0 0.5
2.0 0.0 1.0
-1.0 1.0 1.0
-1
```

1.4 Sample Output

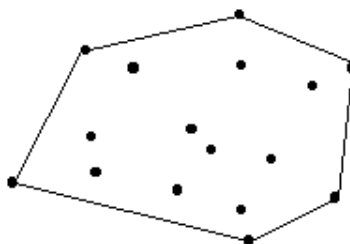
```
The largest component contains 4 rings.
The largest component contains 2 rings.
The largest component contains 3 rings.
```

2 Moth Eradication (218)

Time limit : 3.000 seconds.

Entomologists in the Northeast have set out traps to determine the influx of Joliet moths into the area. They plan to study eradication programs that have some potential to control the spread of the moth population.

The study calls for organizing the traps in which moths have been caught into compact regions, which will then be used to test each eradication program. A region is defined as the polygon with the minimum length perimeter that can enclose all traps within that region. For example, the traps (represented by dots) of a particular region and its associated polygon are illustrated below.



You must write a program that can take as input the locations of traps in a region and output the locations of traps that lie on the perimeter of the region as well as the length of the perimeter.

2.1 Input

The input file will contain records of data for several regions. The first line of each record contains the number (an integer) of traps for that region. Subsequent lines of the record contain 2 real numbers that are the x - and y -coordinates of the trap locations. Data within a single record will not be duplicated. End of input is indicated by a region with 0 traps.

2.2 Output

Output for a single region is displayed on at least 3 lines :

First line : The number of the region. (The first record corresponds to **Region #1**, the second region to **Region #2**, etc.)

Next line(s) : A listing of all the points that appear on the perimeter of the region. The points must be identified in the standard form “(x -coordinate, y -coordinate)” rounded to a single decimal place. The starting point for this listing is irrelevant, but the listing must be oriented *clockwise* and *begin and end with the same point*. For collinear points, any order which describes the minimum length perimeter is acceptable.

Last line : The length of the perimeter of the region rounded to 2 decimal places.

One blank line must separate output from consecutive input records.

2.3 Sample Input

```
3
1 2
4 10
5 12.3
6
0 0
1 1
3.1 1.3
3 4.5
6 2.1
```

```
2 -3.2
7
1 0.5
5 0
4 1.5
3 -0.2
2.5 -1.5
0 0
2 2
0
```

2.4 Sample Output

Region #1:
(1.0,2.0)-(4.0,10.0)-(5.0,12.3)-(1.0,2.0)
Perimeter length = 22.10

Region #2:
(0.0,0.0)-(3.0,4.5)-(6.0,2.1)-(2.0,-3.2)-(0.0,0.0)
Perimeter length = 19.66

Region #3:
(0.0,0.0)-(2.0,2.0)-(4.0,1.5)-(5.0,0.0)-(2.5,-1.5)-(0.0,0.0)
Perimeter length = 12.52

3 Dominos (11504)

Time limit : 3.000 seconds.

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line. However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by hand to get the dominos falling again.

Your task is to determine, given the layout of some domino tiles, the minimum number of dominos that must be knocked down by hand in order for all of the dominos to fall.

3.1 Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers, each no larger than 100 000. The first integer n is the number of domino tiles and the second integer m is the number of lines to follow in the test case. The domino tiles are numbered from 1 to n . Each of the following lines contains two integers x and y indicating that if domino number x falls, it will cause domino number y to fall as well.

3.2 Sample Input

```
1
3 2
1 2
2 3
```

3.3 Output Specification

For each test case, output a line containing one integer, the minimum number of dominos that must be knocked over by hand in order for all the dominos to fall.

3.4 Output for Sample Input

```
1
```

4 SMS for Blinds (11602)

Time limit : 3.000 seconds.

Almost all the devices we use are designed for normal people and in most cases those who are not normal adapt to these devices and we praise them a lot for their brilliance. But for disabled people this might not be that pleasant – Imagine that we are living in the land of cockroaches and we have to walk on the walls by wearing super glue shoes because their roads and bridges are designed in that way.



(a) Draft design of a Normal Mobile Phone



(b) Draft design of a mobile phone designed for blind people

So IBM (International Business Mobile) has designed a new set for blind people that have only one large button for typing characters and three other buttons for sending or receiving phones and changing input modes. As there is only one button for typing SMS, therefore one might need to press a button up to 26 times to type a character. So the designers have omitted some characters : for example ‘k’ is replaced with ‘c’, to reduce the total number of alphabets to 19 (including white space which for simplicity is denoted by the character underscore (‘_’)). The following table shows which characters are replaced by what :

a	b	c	d	e	f	g	h	I	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	‘	’	
						g	c								c													

The characters j, k, q, v, w, x, y, z are written using other characters. So the reduced alphabet size becomes 18. So SMS message contain only these 18 different characters and also white space which is denoted with ‘_’ underscore. So all the SMS messages contain total 19 different characters

The order in which the characters are written on the large button denotes how many pushes are required to type a character. Let’s term this ordering as **KEY ORDER**. For example if the characters are written in order “_abcdilefmnoprghstu”, then to type ‘c’ four presses are required, to type ‘r’ 14 presses are required and so on. And to type “abacus” total $(2+3+2+4+19+17) = 47$ presses are required. Given the **KEY ORDER** and the SMS to type it is very easy to find out the total number of presses to type that message. But in this problem you will be given the SMS to be typed and total number of presses needed to type that SMS. From this information you will have to find the **KEY ORDER**.

4.1 Input

The input file contains total 1200 sets of inputs. The description of each set is given below :

First line of the input for each set contains a string, which is the SMS to be typed. This string contains only 19 different characters and has a maximum length 1000. The next line contains an integer C which denotes the total number of presses to type that string. The characters in the string are mostly randomly generated. But this string can only contain characters present in the string “_abcdefghijklmnopqrstuvwxyz”.

Input is terminated by a line containing a single “*”.

4.2 Output

For each set of input produce one line of output. This line contains a string which denotes the KEY ORDER that will ensure that the typing cost of the given SMS message is equal to C . If there is more than one possible ordering, any one will do. There will be no such input for which a ordering is not found and this string should contain exactly 19 characters (only allowed characters are the characters in the string “_abcdefghijklmnopqrstuvwxyz”) and these characters should be different (No character should be present twice). Note that it may be hard to find a strategy that finds solution for all possible inputs, so if your output formatting is right (String length 19 and all characters different and from the string “_abcdefghijklmnopqrstuvwxyz”) but your cost is wrong for at most 10 cases (of the 1200 cases), then your solution will be accepted.

4.3 Sample Input

```
tsnobfsr_emlllla_dtuangtcrhgpguthngrff
366
aasuufoisuflhdeihdauhnhiatlclusion
353
o_nmhfulhfmihl_febdiaommff_
304
*
```

4.4 Output for Sample Input

```
grudefpin_hbtamscllo
uperbtsaclf_gdonhmi
hsugtrd_pibfnolaemc
```

5 Colossal Fibonacci Numbers! (11582)

Time limit : 3.000 seconds.

The i 'th Fibonacci number $f(i)$ is recursively defined in the following way :

- $f(0) = 0$ and $f(1) = 1$
- $f(i + 2) = f(i + 1) + f(i)$ for every $i \geq 0$

Your task is to compute some values of this sequence.

Input begins with an integer $t \leq 10,000$, the number of test cases. Each test case consists of three integers a, b, n where $0 \leq a, b < 264$ (a and b will not both be zero) and $1 \leq n \leq 1000$.

For each test case, output a single line containing the remainder of $f(ab)$ upon division by n .

5.1 Sample input

```
3
1 1 2
2 3 1000
18446744073709551615 18446744073709551615 1000
```

5.2 Sample output

```
1
21
250
```

6 New Rule in Euphomia (10742)

Time limit : 3.000 seconds.

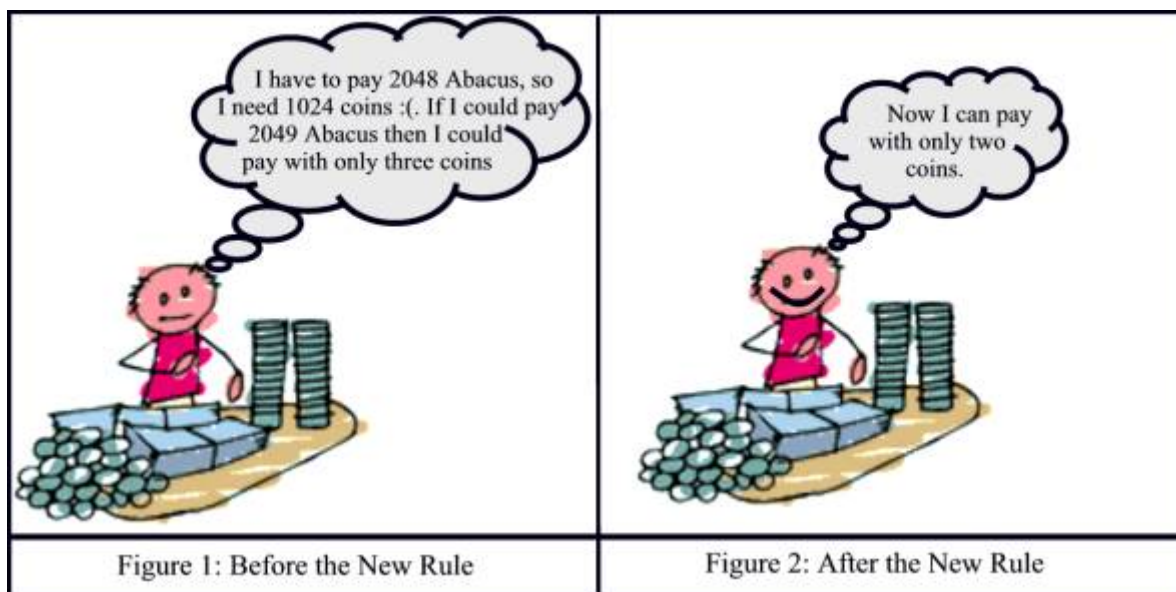
The ancient country of Euphomia had some strange rules regarding payment of prices and values of coins. When King Albert ruled this country the following payment rules were in place :

- The name of the currency was Abacus.
- Only coins were used to pay any certain amount.
- The coins had integer values, *e.g.*, There is no coin with value 1.5 Abacus. And the amount to be paid was also always integer.
- There was no 1 Abacus coin.
- Payments of any amount had to be made with coins of the same value. For example when one paid 9 Abacus he paid it with three 3-Abacus coins.

So coins were made so that any one could pay any amount within **2-1000000** abacuses following the above rules. The coins, which were not required, were not made. For example there were no 6-Abacus coin because one could pay 6 Abacus as (3+3) Abacus. The problem with the above rule was that often one had to use many coins to pay a particular amounts. For example to pay the amount 2048 Abacus one had to use 1024 2-Abacus coins which is quite terrible. Therefore, when king Lucas captured this country he made unlimited coins of value 1 Abacus and added the following rule :

“One can only use exactly two different coins made during Albert’s regime. In addition with these coins one can also use the 1-Abacus coins at any number. However one cannot use only the 1-Abacus coins to make payment.”

So in this new rule paying 1 Abacus is still impossible but rule d and e of Albert’s regime is now obsolete. With this new rule in place your job is to find out in how many ways one could pay a certain amount n .



For example one can now pay 10-Abacus in the following five ways :

- $3 + 2 + 1 + 1 + 1 + 1 + 1$
- $7 + 3$
- $5 + 2 + 1 + 1 + 1$
- $7 + 2 + 1$
- $5 + 3 + 1 + 1$

6.1 Input

The input file contains less than 600 lines of inputs. Each line of the input file contains one value of n ($0 < n \leq 1000000$), which indicates the amount to be paid. Input is terminated by a line where $n = 0$. Obviously this line should not be processed.

6.2 Output

For each line of input except the last one produce one line of output, which contains the serial of output followed by an integer which indicates in how many ways one can pay n abacus with the new rules in place. Look at the output for sample input for details.

6.3 Sample Input

```
10
996542
24
129
0
```

6.4 Output for Sample Input

```
Case 1: 5
Case 2: 1661327749
Case 3: 22
Case 4: 296
```

7 Parentheses Balance (673)

Time limit : 3.000 seconds.

You are given a string consisting of parentheses () and []. A string of this type is said to be correct :

- (a) if it is the empty string
- (b) if A and B are correct, AB is correct,
- (c) if A is correct, (A) and [A] is correct.

Write a program that takes a sequence of strings of this type and check their correctness. Your program can assume that the maximum string length is 128.

7.1 Input

The file contains a positive integer n and a sequence of n strings of parentheses () and [], one string a line.

7.2 Output

A sequence of Yes or No on the output file.

7.3 Sample Input

```
3
([])
(([]))
([()])
```

7.4 Sample Output

```
Yes
No
Yes
```