



3003 – Jelly

Oceania – South Pacific – 2004/2005

A local school provides jelly for their pupils every day, and the school staff are very careful to see that each child has exactly the same amount.

The jelly is prepared the previous day; the liquid jelly is poured into rectangular sided moulds, one mould per child, and then put in the fridge where it sets. The moulds may differ by the length and width of their sides but are filled to different heights so that they all have the same volume; length, width, and height are always integer numbers.

Unfortunately, one of the cleaners loves practical jokes! Whenever he can, before the jelly has set, he tips liquid jelly from one of the moulds into another. He is happy if he succeeds just once and doesn't repeat the joke with other moulds.

Your task is to help the school staff by preparing a report for them. They need to know who has lost jelly and who has gained it so that they can correct matters before the children arrive.

Input

The input consists of one or more scenarios. Each scenario begins with a single integer n , $1 \leq n \leq 100$, representing the number of children for whom jelly was prepared. Following this are n lines, each line representing one child. The data for a child consists of the child's name and 3 integer numbers in the range 1 to 100, respectively representing the length, width and height of the jelly in that child's mould, all separated by single spaces. A child's name consists of a sequence of 1 up to 10 letters (upper and/or lower case), and no two children have the same name. A single 0 on a line by itself marks the end of input.

Output

Your report consists of one line of text per scenario. If the cleaner did not manage to transfer any jelly before it set, your report must say

```
No child has lost jelly.
```

If the cleaner did manage to transfer jelly, your report must be of the form

```
ChildA has lost jelly to ChildB.
```

where *ChildA* is the actual name of the child that has lost jelly and *ChildB* is the actual name of the child that has gained jelly.

Sample Input

```
3
Joe 10 10 2
Susan 10 5 4
Bill 5 5 8
4
Zoe 10 2 2
```

Lee 6 5 2
Alan 5 4 4
Tommy 12 5 1
0

Sample Output

No child has lost jelly.
Zoe has lost jelly to Alan.

South Pacific 2004–2005



3004 – Change

Oceania – South Pacific – 2004/2005

A small local shop is having a problem because the assistants find it hard to work out how much change to give to customers. You have been asked to help them by writing a program that does all the work!

Notes and coins available are as follows:

Notes: \$20, \$10, \$5, \$2, \$1.

Coins: 50c, 20c, 10c, 5c.

As the smallest coin available is 5c, the cost of the purchase may need to be rounded to the nearest 5c, using the so-called Swedish rounding method. The rules for rounding are as follows:

1 or 2 cents - rounded down to 0.

3 or 4 cents - rounded up to 5.

6 or 7 cents - rounded down to 5.

8 or 9 cents - rounded up to 10.

The program must work out the change required and specify the notes and coins to use. In each case, the smallest possible number of notes and coins must be used.

Input

Each line of input will represent a single transaction, and will contain 2 decimal numbers in the range 0.05 to 1000.00, each with two digits after the decimal point, and separated by a single space. The first number is the cost of a purchase, the second the amount the customer offers at the till. As mentioned, the cost of the purchase may need to be rounded, and, of course, the amount offered by the customer is a multiple of 5 cents. A line consisting of two 0.00 numbers marks the end of the input.

Output

Your program must output one line for each transaction. Where the amount of money offered by the customer is not enough to cover the rounded purchase price, your program must output

Not enough money offered.

Where the amount of money offered by the customer is exactly the rounded purchase price, your program must output

Exact amount.

In all other cases output the sequence describing the change. Each sequence item starts with a note or coin value in the format described earlier (e.g., \$2 or 10c), followed by a multiplication sign (i.e., an asterisk, '*')

and ends with a repetition count (a number ≥ 1). Items are listed in order of decreasing values and are separated by single spaces.

Sample Input

```
20.03 20.00
20.07 20.05
20.08 25.00
0.09 0.10
0.00 0.00
```

Sample Output

```
Not enough money offered.
Exact amount.
$2*2 50c*1 20c*2
Exact amount.
```

South Pacific 2004–2005



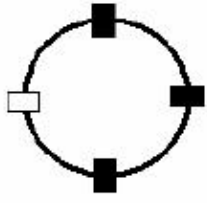
3005 – Necklaces

Oceania – South Pacific – 2004/2005

In the Macmahara Desert, a group of explorers have found the remains of an interesting tribe of people, the Yacms. A number of necklaces have been found which, according to local history, the Yacms used to represent numbers. Each necklace consists of a continuous wire on which are strung a number of beads. As the beads are of only two colours (black and white), it is clear that the necklaces were used to represent binary numbers.

Unfortunately, nobody knows whether the black represented 1 and the white 0 or the other way around. It is also not clear where the number started, or even in which direction it was to be read!

A simple necklace like the one below could represent several numbers.

	Black as 1		White as 1	
	Read clockwise	Read anticlockwise	Read clockwise	Read anticlockwise
	BBBW = 14	WBBB = 7	BBBW = 1	WBBB = 8
	BBWB = 13	BBBW = 14	BBWB = 2	BBBW = 1
	BWBB = 11	BBWB = 13	BWBB = 4	BBWB = 2
	WBBB = 7	BWBB = 11	WBBB = 8	BWBB = 4

What we can say is that the lowest number it can represent is 1, the highest 14. In this simple case the lowest and the highest numbers are the same whether we read the necklace clockwise or anticlockwise. However, there are complex cases when the two reading directions also give different lowest and highest numbers. As you can verify, the last necklace listed in the sample input is indeed such a case.

Your task is to find out the lowest number and the highest number that a given necklace could possibly represent.

Input

Input will consist of a number of lines, each representing one necklace. Each necklace will be represented by a sequence of 1 to 30 letters, upper case 'B' to represent a black bead, upper case 'W' a white bead. The word 'END' will mark the end of input.

Output

For each input line, output 2 integers (in base 10), separated by a single space. The first is the lowest number that the necklace could represent, the second the highest number.

Sample Input

```
B  
BW  
BBWB  
BBBW  
WBBWB  
BBBWBWW  
END
```

Sample Output

```
0 1  
1 2  
1 14  
1 14  
5 26  
11 116
```

South Pacific 2004–2005



3006 – Graphics

Oceania – South Pacific – 2004/2005

In some graphics applications the screen is divided into 4 areas numbered 1, 2, 3, 4, with each area recursively divided in the same way, to an arbitrary depth. Using this scheme, each cell on the screen can be uniquely identified by a string of digits in the range 1–4, as in the following diagrams:

You are given a sequence of digits in the range 1–4 identifying an initial cell and a sequence of moves of the form U(up), D(down), L(left), R(right). Your task is to identify the destination of this move sequence, also as a sequence of digits in the range 1–4, or to write OUT if the trajectory gets beyond the screen borders.

Input

Input will consist of a series of scenarios, each specified by two lines of input:

- The first line consists of a string of 1 to 255 digits that specifies an initial cell, where each digit is in the range 1–4.
- The second line consists of a string of 1 to 255 letters that specifies the series of moves, where each letter is one of 'U', 'D', 'L', 'R'.

The end of the input is indicated by a line that consists of the word 'END'.

Output

Output for each scenario should consist of a single line that specifies the final cell, as a sequence of digits in the range 1–4, or the word 'OUT'.

Sample Input

```
1
RD
31
ULDRR
421
LLDRRRU
244444
DL
END
```

Sample Output

```
3
32
OUT
422222
```

South Pacific 2004–2005



3007 – Shopping

Oceania – South Pacific – 2004/2005

The local supermarket has survived with one checkout for many years. Yet, recently the supermarket has become much busier. The manager is planning to add more checkouts but this will take some time.

As an interim measure the manager aims to minimize the total time the customers wait in the queue for the checkout. More precisely, this total waiting time is defined as the sum of all customer waiting times, where a customer is considered to be waiting when in the queue, but not considered to be waiting when being served at the checkout. The shop will stay open until all the customers have been served.

The manager has decided to experiment with a more flexible checkout service policy. Under this new policy:

- A waiting customer and even a newly arriving customer can be directed to the checkout regardless of her current position in the waiting queue.
- A customer can be returned to the queue part way through being served at the checkout. When that customer later returns to the checkout only the remaining time to complete her checkout procedure is required.
- No extra time is required to pick a customer from the queue or to put a customer back onto the queue.

Your task is to work out the minimum total waiting time as defined above.

Input

The first line contains the number of scenarios you will be given.

Each scenario begins with an integer n which is the number of customers where $1 \leq n \leq 400$. Then the next n lines contain the information for customers labelled 1 to n . Each of these lines contains two numbers i and j separated by a single space, where $0 \leq i \leq 36000$ and $1 \leq j \leq 600$, with the following meaning:

- i represents the time the customer arrives at the checkout with her load of shopping, this is given as the number of seconds from the time the shop opened.
- j represents the time required to scan and pay for her shopping.

Output

For each scenario output a line consisting of the number of customers and the minimum total waiting time in seconds, separated by a single space.

Sample Input

```
3
4
2000 500
200 300
1500 200
2300 400
4
100 100
100 100
100 200
100 200
```


3
100 500
100 100
200 100

Sample Output

4 200
4 700
3 200

South Pacific 2004–2005



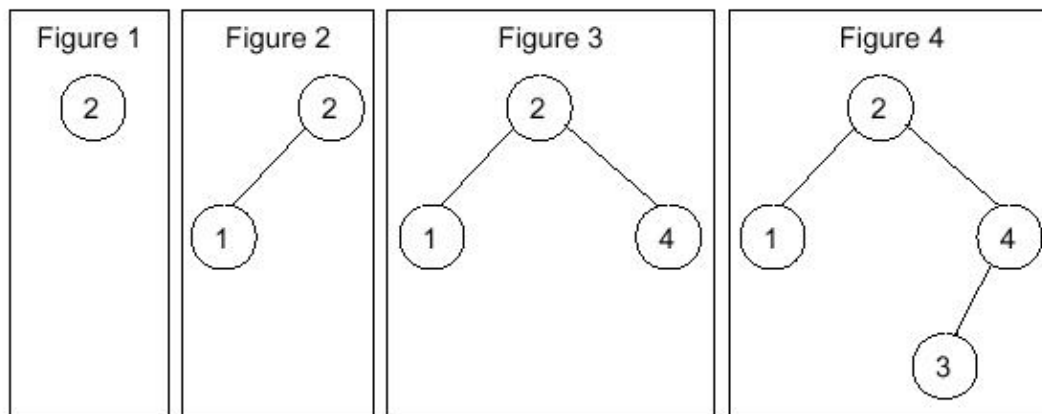
3008 – Trees

Oceania – South Pacific – 2004/2005

A Binary Search Tree (BST) is a binary tree where every node has a value and the tree is arranged so that, for any node all the values in its left subtree are less than the node's value, and all the values in its right subtree are greater than the node's value.

To build a BST from a sequence of distinct integers the following procedure is used. The first integer becomes the root of the tree. Then the second integer in the sequence is considered. If it is less than the value of the root node then it becomes the left child. Otherwise, it becomes the right child. Subsequent items in the sequence move down either left or right in the same fashion depending on the comparison against earlier nodes, starting at the root and progressing down the tree. The new node is inserted (as a leaf) into the partially constructed tree when it reaches a missing subtree in the particular direction of travel.

For example, a BST generated by the sequence 2, 1, 4, 3 is built up as shown below as the numbers are inserted.



The tree of Figure 4 can also be generated by two other sequences: 2, 4, 3, 1 and 2, 4, 1, 3.

Such sequences can be compared according to a lexicographic order, where the comparison proceeds left-to-right and items at corresponding positions are compared using their numerical values. The result is shown below, using "<" to denote this lexicographic order between sequences:

$2, 1, 4, 3 < 2, 4, 1, 3 < 2, 4, 3, 1.$

For the tree in Figure 4 the lexicographically least sequence is 2, 1, 4, 3.

Write a program that will read a representation of a tree and will generate the lexicographically least sequence of distinct positive integers that will generate that tree. Note that for a tree with n nodes this sequence will be a permutation of the numbers from 1 to n .

For the input to our problem we represent (recursively) the structure of a tree as follows:

1. A single node (a leaf) is a tree:

◆ $()$
2. If T_L and T_R are trees then the following are also trees:

- ◆ (T_L)
- ◆ $(,T_R)$
- ◆ (T_L,T_R)

Input

Input will consist of a sequence of lines. Each line will consist of up to 250 characters and will contain the specification of a single tree according to the above definition (with no intervening spaces). The sequence will be terminated by a tree consisting of a single node $()$. This line should not be processed.

Output

Output will be a sequence of lines, one for each line in the input. Each line will consist of the required permutation of the integers from 1 to n (where n is the number of nodes in the tree) separated by single spaces.

Sample Input

```
((), ((), ))  
((((), ), ())  
((((), ()), ((), ()))  
(, ())  
((), )  
()
```

Sample Output

```
2 1 4 3  
3 2 1 4  
4 2 1 3 6 5 7  
1 2  
2 1
```

South Pacific 2004–2005



3009 – Monks

Oceania – South Pacific – 2004/2005

The order of Avidly Calculating Monks have their ashram high in the Sierra Nevada mountains east of Silicon Valley. After many years of training, each novice is faced with a final test before he can become a full member of the ACM.

In this test, the novice is led into a room containing three large urns, each containing a number of delicate glass beads. His task is to completely empty one of the urns using the following special procedure. Each day, the novice must select two of the urns, the source, and the destination. He must then carefully move beads from the source urn to the destination urn, never breaking one, until the original contents of the destination urn are doubled. No other beads may be moved.

So for example, if the number of beads in the three urns were respectively

115, 200 and 256 beads

then the novice might choose the second urn as the source and the first as the destination which would result in new contents of

230, 85 and 256 beads

at the end of the first day. Then on the second day the novice might choose the urn with 256 beads as the source, and that with 85 as the destination leaving

230, 170 and 171 beads

at the end of the second day.

If the original contents were 12, 30, and 12 beads, then choosing the first as source and the third as destination would result in 0, 30, and 24 beads and completion of the task.

The chief guru of the ACM always likes to have available a crib sheet indicating how many days the novices should take if they use as few transfers as possible to empty one of the urns. Your task is to provide this crib sheet.

Given a sequence of scenarios, each with a triple of non-negative integers representing the urn contents, determine for each scenario the smallest possible number of days required to empty one of the urns. This problem is guaranteed to have a solution in each case.

Input

Input will consist of a sequence of lines, each line representing a scenario. Each line consists of three integers in the range from 0 to 500 representing the initial urn contents, separated by single spaces. Input is terminated by the line '0 0 0', which is not processed.

Output

For each scenario `` $a b c$ `, output a line of the form

$a b c d$

where d is the smallest possible number of days required to empty an urn beginning from contents a, b , and c .

Sample Input

```
0 3 5
5 0 3
1 1 1
2 3 4
12 3 8
0 0 0
```

Sample Output

```
0 3 5 0
5 0 3 0
1 1 1 1
2 3 4 2
12 3 8 5
```

South Pacific 2004–2005



3010 – Squares

Oceania – South Pacific – 2004/2005

Consider a 3 by 3 arrangement of the digits 1 to 9, as illustrated in the following diagram:

Figure 1

The arrangement can be modified by rotating any of the 2-by-2 groups in the corners, either clockwise or anticlockwise. Thus if the top-right corner of the above arrangement is rotated anticlockwise, the result is the following arrangement:

Figure 2

A magic square is an n -by- n arrangement of numbers, such that the sum of the numbers in each row, column, and diagonal is the same. For example, the following diagram illustrates one possible 3-by-3 magic square for the numbers 1 to 9:

Figure 3

Your task is to determine the minimum number of moves to transform a given digit arrangement into a magic square.

For example, the magic square in Figure 3 can be obtained from the arrangement illustrated in Figure 2 by one clockwise rotation of the top-left corner. Thus the arrangement given in Figure 1 can be transformed into a magic square in 2 moves (and, as you can verify, no shorter sequences of moves would suffice).

Input

Input will consist of a series of lines, each specifying an initial arrangement of the digits 1 to 9, listed in row-by-row order.

The end of the input is indicated by a line that consists of the word 'END'.

Output

Output for each arrangement should consist of either:

- the minimum number of moves followed by a single space and then the word 'moves', or
- the word 'IMPOSSIBLE', if it is not possible to achieve a magic square arrangement.

Sample Input

```
135876492
438975261
672159834
129764583
END
```

Sample Output

2 moves
1 moves
0 moves
4 moves

South Pacific 2004–2005



3011 – Colours

Oceania – South Pacific – 2004/2005

A manager for a toy company wants to reduce the cost of manufacturing their line of toys. Briefly, the toys are created by robots that operate on assembly tracks by adding and linking track components into modules and by merging existing modules into more complex modules. Components can be either active or inactive. At any moment there is exactly one active component per each module and track; and this is the only component that can be linked or merged on that track for that module.

The new budget for this company will only support components which consist of three colours and adjacent components must have different colours. Your job is to decide which of the current toys in the inventory can be produced with this colouring limitation.

The company uses the following BNF formalism to describe more precisely the blueprints of its toys:

1. $\langle \text{toy} \rangle ::= \langle \text{last-track} \rangle \langle \text{module} \rangle$

The current $\langle \text{toy} \rangle$ consists of a main $\langle \text{module} \rangle$. $\langle \text{last-track} \rangle$ gives the number of the last track used to build the current $\langle \text{toy} \rangle$; the tracks are numbered from 0 to $\langle \text{last-track} \rangle$.

2. $\langle \text{module} \rangle ::= \langle ' \langle \text{operator-sequence} \rangle ' \rangle$

$\langle \text{module} \rangle$ represents a simple module that only contains operators.

The operators given by the $\langle \text{operator-sequence} \rangle$ are processed in a left-to-right order.

This $\langle \text{module} \rangle$ starts with empty tracks and then automatically adds one active component on each of the available tracks.

3. $\langle \text{merged-module} \rangle ::= \langle ' \langle \text{module} \rangle_1 \langle \text{module} \rangle_2 ' \rangle$

This is a merge operation that builds a complex $\langle \text{merged-module} \rangle$ by merging the active components of $\langle \text{module} \rangle_1$ and $\langle \text{module} \rangle_2$, after both modules are completely built.

4. $\langle \text{module} \rangle ::= \langle ' \langle \text{merged-module} \rangle \langle \text{operator-sequence} \rangle ' \rangle$

The components of the $\langle \text{merged-module} \rangle$ remain on the tracks and its active components are further worked upon by the operators given by $\langle \text{operator-sequence} \rangle$

5. $\langle \text{operator-sequence} \rangle ::= \langle \lambda \mid \langle \text{operator} \rangle \langle \text{operator-sequence} \rangle \rangle$

6. $\langle \text{operator} \rangle ::= \langle \text{node-operator} \rangle \mid \langle \text{edge-operator} \rangle$

7. $\langle \text{node-operator} \rangle ::= \langle \text{track-number} \rangle$

A $\langle \text{node-operator} \rangle$ adds an active component on the specified $\langle \text{track-number} \rangle$ for the current module and the previously active component on that track becomes inactive.

8. $\langle \text{edge-operator} \rangle ::= \langle \text{track-number-pair} \rangle$

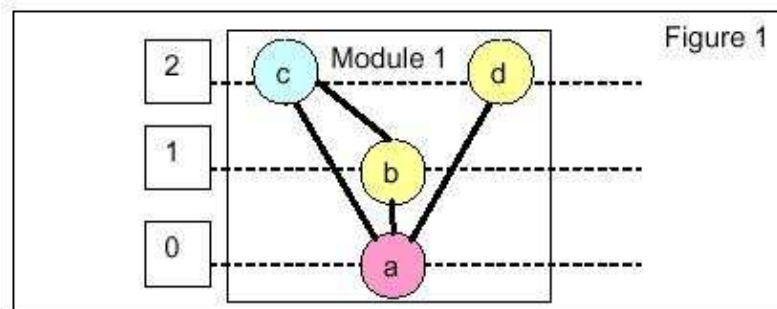
An $\langle \text{edge-operator} \rangle$ links the active components of the two track-numbers.

The following examples show several simple toy blueprints; in the figures tracks are represented as horizontal dotted lines, components as circles, links as full lines, and the time axis flows from left to right.

EXAMPLE 1

Figure 1 depicts a 3-colourable toy that can be built using the following blueprint:

2 (20 10 21 2 20)



There are 3 tracks numbered 0, 1, 2. The toy consists of a single module containing 4 components labelled a , b , c , d , linked by the lines $c-a$, $b-a$, $c-b$, $d-a$. To build this toy the robot will execute in order the following operations:

1. Add a on track 0, b on track 1, c on track 2. At this stage a , b , c are the active components.
2. Make the links $c-a$, $b-a$, $c-b$.
3. Add d on track 2, which makes d active and inactivates c .
4. Make the link $d-a$. At this stage a , b , d are the active components.

Note that the same toy can also be built using several other blueprints, such as the following two:

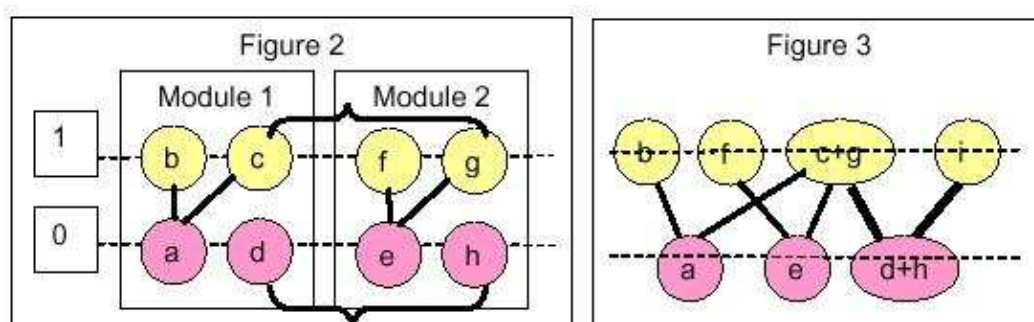
2 (10 20 21 2 20)

2 (((20 10) (21)) 2 20)

EXAMPLE 2

Figures 2 and 3 illustrate a sequence of operations involving a merge, for another 3-colourable (in fact even 2-colourable) toy that can be built as specified by the following blueprint:

1 (((10 1 10 0) (10 1 10 0)) 10 1 10)



1. First, module 1 is built:

- a. Add a on track 0 and b on track 1.
- b. Link $b-a$.

- c. Add c on track 1.
 - d. Link $c-a$.
 - e. Add d on track 0. c, d are now the active components of module 1.
2. Secondly, module 2 is built using similar operations. g, h are now the active components of module 2.
 3. Thirdly, modules 1 and 2 are merged together, which means that active components of each track are identified, i.e., $c = g, d = h$.

The snapshot of Figure 2 illustrates this moment, with braces showing component identification.

4. Fourthly, the just merged components are linked, i.e., $(c + g)-(d + h)$

$(c + g)-(d + h)$ are now the active components of the merged module 1+2.

5. Lastly, i is added to track 1 and linked with $(d + h)$.

The final result is depicted in Figure 3 and links made after the merging are depicted with double lines. At the end, i and $(d + h)$ are the active components of the main module.

Input

The input will consist of a sequence of toy blueprints, one per line of at most 250 characters. Each toy blueprint contains a sequence of tokens separated by single spaces, and conforming to the BNF rules stated earlier. The first token is a positive integer t , $0 \leq t \leq 6$, denoting the maximum track number for the robot's arms to grab (i.e., there are $t+1$ current components for the robot).

The interpretations for the remaining tokens are given in the next table.

The input will be terminated by a toy description with $t = 0$, which is not processed.

Output

The required output is a line of the form `Toy # : ?', where `#' denotes the toy sequence number starting at 1 and ? is either `Yes' or `No' depending whether the toy can be properly built using at most 3 colours.

Sample Input

```
2 ( 20 10 21 2 20 )
1 ( ( ( 10 1 10 0 ) ( 10 1 10 0 ) ) 10 1 10 )
3 ( 32 31 20 21 10 0 10 30 20 )
2 ( ( ( 10 1 10 21 1 21 10 ) ( 21 ) ) 0 10 20 )
0 ( )
```

Sample Output

```
Toy 1: Yes
Toy 2: Yes
Toy 3: No
Toy 4: Yes
```