Problem A

Coconuts, Revisited

The short story titled *Coconuts*, by Ben Ames Williams, appeared in the *Saturday Evening Post* on October 9, 1926. The story tells about five men and a monkey who were shipwrecked on an island. They spent the first night gathering coconuts. During the night, one man woke up and decided to take his share of the coconuts. He divided them into five piles. One coconut was left over so he gave it to the monkey, then hid his share and went back to sheep.

Soon a second man woke up and did the same thing. After dividing the coconuts into five piles, one coconut was left over which he gave to the monkey. He then hid his share and went back to bed. The third, fourth, and fifth man followed exactly the same procedure. The next morning, after they all woke up, they divided the remaining coconuts into five equal shares. This time no coconuts were left over.

An obvious question is "how many coconuts did they originally gather?" There are an infinite number of answers, but the lowest of these is 3,121. But that's not our problem here.

Suppose we turn the problem around. If we know the number of coconuts that were gathered, what is the maximum number of persons (and one monkey) that could have been shipwrecked if the same procedure could occur?

Input

The input will consist of a sequence of integers, each representing the number of coconuts gathered by a group of persons (and a monkey) that were shipwrecked. The sequence will be followed by a negative number.

Output

For each number of coconuts, determine the largest number of persons who could have participated in the procedure described above. Display the results similar to the manner shown below, in the Expected Output. There may be no solution for some of the input cases; if so, state that observation.

Sample Input

```
25
30
3121
-1
```

Expected Output

```
25 coconuts, 3 persons and 1 monkey
30 coconuts, no solution
3121 coconuts, 5 persons and 1 monkey
```

Problem B

Nonstop Travel

Fast Phil works the late shift and leaves his company's parking lot at precisely 2:00 AM every morning. His route home is by a straight road which has one or more traffic signals. Phil has always wondered if, given the locations and cycles of each of the traffic signals, are there velocities he can travel home without ever having to speed up or slow down on account of a red light. You are to write a program to satisfy his curiosity.

Your program should find all integer speeds (in miles per hour) which can be used for Phil's trip home. Each speed is a rate (in miles per hour) he can maintain the moment he leaves the parking lot at 2:00 AM until he arrives home (we assume Phil has a long driveway in which to decelerate) such that he never passes through a red signal. He is allowed to pass throgh a signal at the exact moment it turns from yellow to red, or at the exact moment a red signal turns green. Since Phil is a relatively law-abiding citizen, you need only consider speeds less than or equal to 60 mph. Likewise, Phil isn't interested in travelling too slowly, so you should not consider speeds lower than 30 mph.

Input

Input will consist of one or more sets of data describing a set of traffic signals, followed by the integer -1. The first integer in each set will contain the value *N* (specifying the number of traffic signals). *N* will be no larger than 6. This value will be followed by *N* sets of numbers, each containing values (in order) for *L*, *G*, *Y* and *R*. *L* is a positive real number indicating the location of a traffic signal, in miles, from the parking lot. *G*, *Y* and *R* are the lengths of time (in seconds) of the green, yellow, and red periods of the corresponding traffic signal's cycle. Phil has learned from an informant in the Highway Department that all *N* traffic signals start their green period precisely at 2:00 AM.

Output

Output should consist of the input case number (starting with 1) followed by a list of all valid integer speeds Phil may drive to avoid the red signals. Consecutive integer speeds should be specified in interval notation of the form `L-H`, where `L` and `H` are the lowest and highest speeds for the interval. Intervals of the form `L-L` (that is, an interval of length 1) shold just be written as `L`. Intervals should be separated by commas. If there are no valid speeds, you program should display the phrase *`No acceptable speeds.`* The Expected Output below illustrates this format.

Sample Input

```
1
5.5  40 8 25
3
10.7  10 2 75
12.5  12 5 57
17.93 15 4 67
-1
```

Expected Output

```
Case 1: 30, 32-33, 36-38, 41-45, 48-54, 59-60
Case 2: No acceptable speeds.
```

Problem C

Image Perimeters

Technicians in a pathology lab analyze digitized images of slides. Objects on a slide are selected for analysis by a mouse click on the object. The perimeter of the boundary of an object is one useful measure. Your task is to determine this perimeter for selected objects.

The digitized slides will be represented by a rectangular grid of periods, '.', indicating empty space, and the capital letter 'X', indicating part of an object. Simple examples are

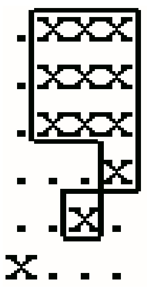
```
XX   Grid 1       .XXX   Grid 2
XX                .XXX
                  .XXX
                  ...X
                  ..X.
                  X...
```

An X in a grid square indicates that the entire grid square, including its boundaries, lies in some object. The X in the center of the grid below is *adjacent* to the X in any of the 8 positions around it. The grid squares for any two adjacent X's overlap on an edge or corner, so they are connected.

```
XXX
XXX      Central X and adjacent X's
XXX
```

An object consists of the grid squares of all X's that can be linked to one another through a sequence of adjacent X's. In Grid 1, the whole grid is filled by one object. In Grid 2 there are two objects. One object contains only the lower left grid square. The remaining X's belong to the other object.

The technician will always click on an X, selecting the object containing that X. The coordinates of the click are recorded. Rows and columns are numbered starting from 1 in the upper left hand corner. The technician could select the object in Grid 1 by clicking on row 2 and column 2. The larger object in Grid 2 could be selected by clicking on row 2, column 3. The click could not be on row 4, column 3.



One useful statistic is the perimeter of the object. Assume each X corresponds to a square one unit on each side. Hence the object in Grid 1 has perimeter 8 (2 on each of four sides). The perimeter for the larger object in Grid 2 is illustrated in the figure at the left. The length is 18.

Objects will not contain any totally enclosed holes, so the leftmost grid patterns shown below could *NOT* appear. The variations on the right could appear:

```
Impossible   Possible

XXXX         XXXX   XXXX   XXXX
X..X         XXXX   X...   X...
XX.X         XXXX   XX.X   XX.X
XXXX         XXXX   XXXX   XX.X

.....        .....  .....  .....
..X..        ..X..  ..X..  ..X..
.X.X.        .XXX.  .X...  .....
..X..        ..X..  ..X..  ..X..
.....        .....  .....  .....
```

The input will contain one or more grids. Each grid is preceded by a line containing the number of rows and columns in the grid and the row and column of the mouse click. All numbers are in the range 1-20. The rows of the grid follow, starting on the next line, consisting of `'.'` and 'X' characters.

The end of the input is indicated by a line containing four zeros. The numbers on any one line are separated by blanks. The grid rows contain no blanks.

For each grid in the input, the output contains a single line with the perimeter of the specified object.

Example input:	**Example output:**

```
2 2 2 2
XX
XX
6 4 2 3
.XXX
.XXX
.XXX
...X
..X.
X...
5 6 1 3
.XXXX.
X....X
..XX.X
.X...X
..XXX.
7 7 2 6
XXXXXXX
XX...XX
X..X..X
X..X...
X..X..X
X.....X
XXXXXXX
7 7 4 4
XXXXXXX
XX...XX
X..X..X
X..X...
X..X..X
X.....X
XXXXXXX
0 0 0 0
```
	```
	8
	18
	40
	48
	8
	```

Problem D: Falling Leaves

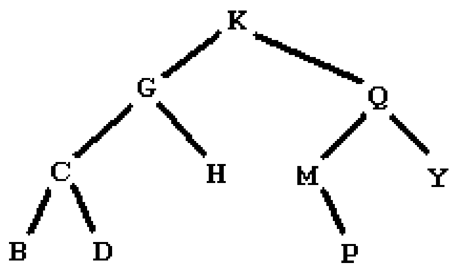


Figure 1

Figure 1 shows a graphical representation of a binary tree of letters. People familiar with binary trees can skip over the definitions of a binary tree of letters, leaves of a binary tree, and a binary search tree of letters, and go right to **The problem**.

A *binary tree of letters* may be one of two things:

1. It may be empty.
2. It may have a root *node*. A node has a letter as data and refers to a left and a right subtree. The left and right subtrees are also binary trees of letters.

In the *graphical* representation of a binary tree of letters:

1. Empty trees are omitted completely.
2. Each node is indicated by
   - Its letter data,
   - A line segment down to the left to the left subtree, if the left subtree is nonempty,
   - A line segment down to the right to the right subtree, if the right subtree is nonempty.

A *leaf* in a binary tree is a node whose subtrees are both empty. In the example in Figure 1, this would be the five nodes with data `B`, `D`, `H`, `P`, and `Y`.

The *preorder traversal of a tree of letters* satisfies the defining properties:

1. If the tree is empty, then the preorder traversal is empty.
2. If the tree is not empty, then the preorder traversal consists of the following, in order
   - The data from the root node,
   - The preorder traversal of the root's left subtree,
   - The preorder traversal of the root's right subtree.

The preorder traversal of the tree in Figure 1 is `KGCBDHQMPY`.

A tree like the one in Figure 1 is also a binary search tree of letters. A *binary search tree of letters* is a binary tree of letters in which each node satisfies:

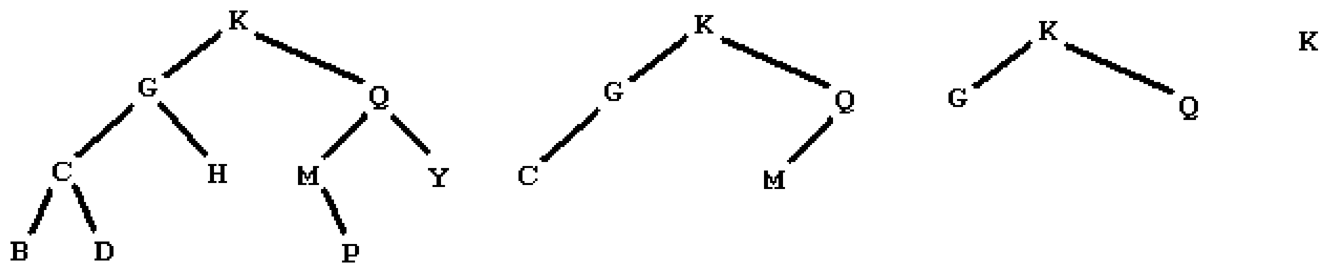
1. The root's data comes later in the alphabet than all the data in the nodes in the left subtree.
2. The root's data comes earlier in the alphabet than all the data in the nodes in the right subtree.

The problem:

Consider the following sequence of operations on a binary search tree of letters

Remove the leaves and list the data removed
Repeat this procedure until the tree is empty

Starting from the tree below on the left, we produce the sequence of trees shown, and then the empty tree



by removing the leaves with data

```
BDHPY
CM
GQ
K
```

Your problem is to start with such a sequence of lines of leaves from a binary search tree of letters and output the preorder traversal of the tree.

The input file will contain one or more data sets. Each data set is a sequence of one or more lines of capital letters. The lines contain the leaves removed from a binary search tree in the stages described above. The letters on a line will be listed in increasing alphabetical order. Data sets are separated by a line containing only an asterisk ('*'). The last data set is followed by a line containing only a dollar sign ('$'). There are no blanks or empty lines in the input.

For each input data set, there is a unique binary search tree that would produce the sequence of leaves. The output is a line containing only the preorder traversal of that tree, with no blanks.

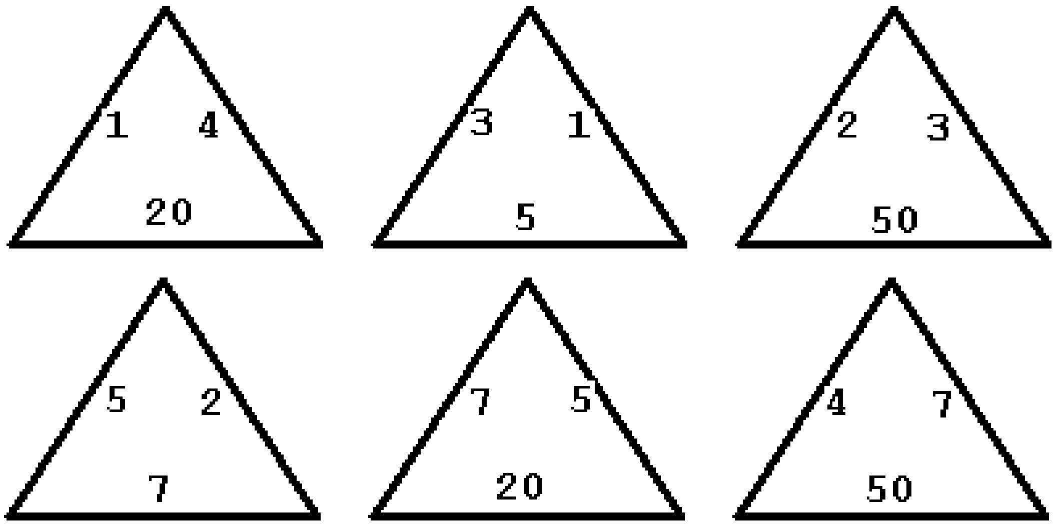
Example input:

```
BDHPY
CM
GQ
K
*
AC
B
$
```

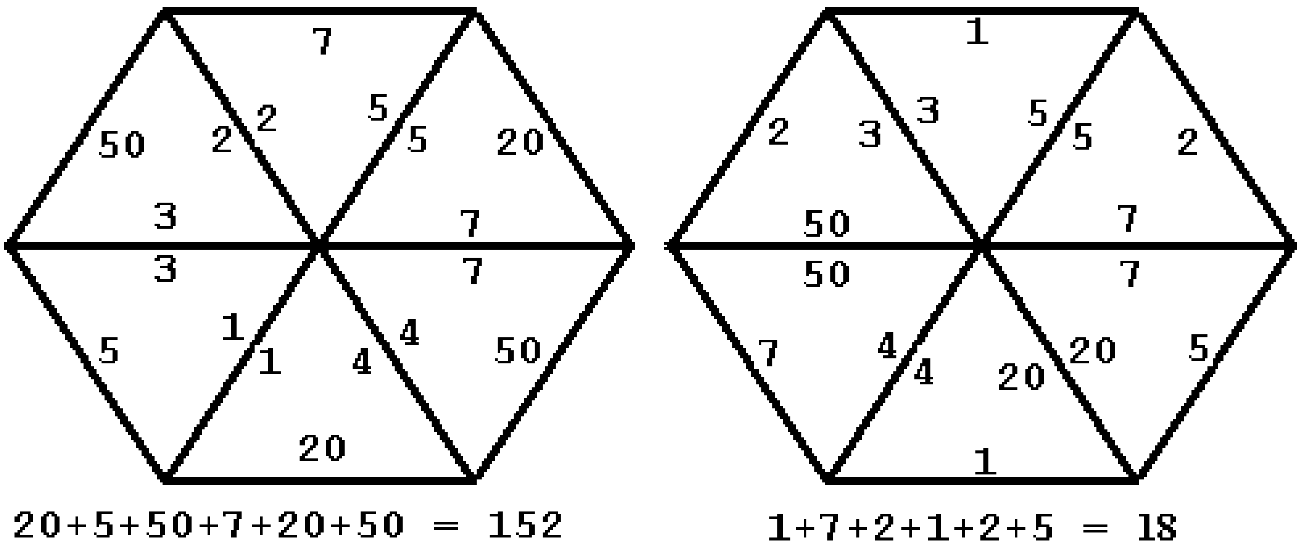
Example output:

```
KGCBDHQMPY
BAC
```

Problem E: The Triangle Game



In the triangle game you start off with six triangles numbered on each edge, as in the example above. You can slide and rotate the triangles so they form a hexagon, but the hexagon is only legal if edges common to two triangles have the same number on them. You may not flip any triangle over. Two legal hexagons formed from the six triangles are illustrated below.



The score for a legal hexagon is the sum of the numbers on the outside six edges.

Your problem is to find the highest score that can be achieved with any six particular triangles.

The input file will contain one or more data sets. Each data set is a sequence of six lines with three integers from 1 to 100 separated by blanks on each line. Each line contains the numbers on the triangles in clockwise order. Data sets are separated by a line containing only an asterisk. The last data set is followed by a line containing only a dollar sign.

For each input data set, the output is a line containing only the word "none" if there are no legal hexagons or the highest score if there is a legal hexagon.

Example input:

```
1 4 20
3 1 5
50 2 3
5 2 7
7 5 20
4 7 50
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
60 6 10
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
10 6 60
$
```

Example output:

```
152
21
none
```