MIT Programming Contest
Team Practice Contest Problems
2003

October 12th, 2003

1 Battle

After Farmer John decided to convert his dairy farm into a slaughterhouse, all of his cows revolted. After a long and intense struggle, the cows have been surrounded by Farmer John's advancing army of rovers. A rover is a doughnut-shaped robot with some radius and exploding range. The cows managed to get a copy of FJ's battle plan, so they know that each rover is going to move to some pre-defined location and fortify at that location.

Fortunately, the cows have a few cannons at their disposal and can fire at the rovers. Each cannon fires in an high parabolic arc, falling on some poor rover. If a rover is hit, it explodes, causing all other rovers within its exploding range to ignite and also explode, which may cause other rovers in its vicinity to explode. For example, if a distance of 3 separates two radius 1 rovers with exploding range 1, then if one of the rovers explodes, then the explosion will just barely reach the other rover, causing it to explode as well.

Unfortunately, the cows don't want to waste all their precious artillery defending themselves. If they wish to have any hope of escaping, they will need to launch a counterattack on FJ after destroying his rovers. So they want to find the minimum number of cannons to fire in order to destroy all of the rovers.

Input

There will be several test cases, each representing a battle scenario.

The first line of each test case will contain a positive integer N ($1 \leq N \leq 1000$), the number of rovers. The subsequent N lines will contain four integers X, Y, R, E, representing the position (X, Y) of the rover, the radius of the rover R, and the exploding range of the rover E.

The input data is terminated by a line that contains one zero, and should not be processed.

Output

The output should contain, for each battle scenario, the minimum number of cannons to fire to destroy all the rovers. Also, list the rovers that they should initially hit. If there are many possible lists of rovers of minimum size, then print out the lexicographically smallest one. Formatting should be as in the following sample output.

Example

Sample Input

```
3
4 7 2 2
8 5 1 0
3 -3 1 1
0
```

Sample Output

```
Battle 1: (2) 1 3
```

2 Bignum Arithmetic

In this problem, you will be concerned with integers with very large numbers of digits. You must write code which will repeatedly accept (until end of file) two lines each containing an unsigned integer, and output the product of the two input unsigned integers. The output must not contain any leading zeros.

You can assume that each integer will contain at most 80 digits. The input ends with an end of file.

Example

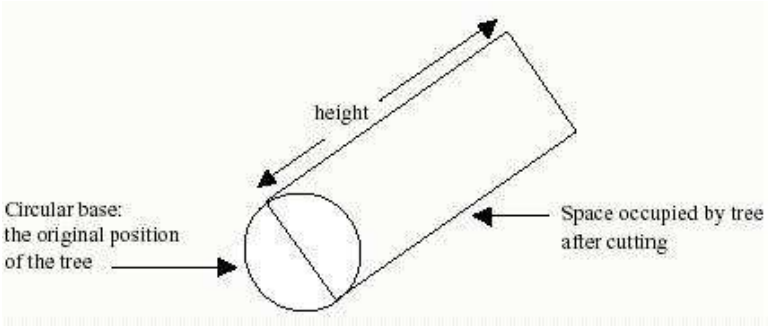
Sample Input	Sample Output
0342 1298 12 3	443916 36

This page intentionally left blank.

3 Can't Cut Down the Forest for the Trees

Once upon a time, in a country far away, there was a king who owned a forest of valuable trees. One day, to deal with a cash flow problem, the king decided to cut down and sell some of his trees. He asked his wizard to find the largest number of trees that could be safely cut down.

All the king's trees stood within a rectangular fence, to protect them from thieves and vandals. Cutting down the trees was difficult, since each tree needed room to fall without hitting and damaging other trees or the fence. Each tree could be trimmed of branches before it was cut. For simplicity, the wizard assumed that when each tree was cut down, it would occupy a rectangular space on the ground, as shown below. One of the sides of the rectangle is a diameter of the original base of the tree. The other dimension of the rectangle is equal to the height of the tree.



Many of the king's trees were located near other trees (that being one of the tell-tale signs of a forest.) The wizard needed to find the maximum number of trees that could be cut down, one after another, in such a way that no fallen tree would touch any other tree or the fence. As soon as each tree falls, it is cut into pieces and carried away so it does not interfere with the next tree to be cut.

Input

The input consists of several test cases each describing a forest. The first line of each description contains five integers, $x_{\min}, y_{\min}, x_{\max}, y_{\max}$, and n. The first four numbers represent the minimal and maximal coordinates of the fence in the x- and y-directions ($x_{\min} < x_{\max}, y_{\min} < y_{\max}$). The fence is rectangular and its sides are parallel to the coordinate axes. The fifth number n represents the number of trees in the forest ($1 < n < 100$).

The next n lines describe the positions and dimensions of the n trees. Each line contains four integers, x_i, y_i, d_i, and h_i, representing the position of the tree s center (x_i, y_i), its base diameter d_i, and its height h_i. No tree bases touch each other, and all the trees are entirely inside the fence, not touching the fence at all.

The input is terminated by a test case with $x_{\min} = y_{\min} = x_{\max} = y_{\max} = n = 0$. This test case should not be processed.

Output

For each test case, first print its number. Then print the maximum number of trees that can be cut down, one after another, such that no fallen tree touches any other tree or the fence. Follow the format in the sample output given below. Print a blank line after each test case.

Sample Input

```
0 0 10 10 3
3 3 2 10
5 5 3 1
2 8 3 9
0 0 0 0 0
```

Sample Output

```
Forest 1
2 tree(s) can be cut
```

4 Design Optimal Bicycle Gear Ratios

As an engineer for the Bicycle Sprint Club, you are responsible for choosing gear ratios for the bikes. The rules of sprint racing say that the bicycle can have only a single gear; that is, one chain connecting a single gearwheel on the pedals to a single gearwheel on the rear wheel. The tire sizes are also fixed. The team physiologist has computed for each member of the team the optimal gear ratio between the number of turns of the pedals to the number of revolutions of the rear wheel. It is your job to translate this single real number into a pair of gears for the bicycle.

The number of teeth on a gearwheel must lie between 16 and 100 due to mechanical constraints. In addition, the pedal gear will always have more teeth, since in sprinting the rear wheel always spins faster than the pedals.

Your input will consist of multiple data sets, one per line. Each set will contain a single real number greater than one and less than six indicating the number of times the rear wheel should spin for each revolution of the pedals.

Your output for each data set should be two integers on a single line. The first integer is the number of gears on the pedal gearwheel and the second integer is the number of gears on the rear wheel gearwheel, such that the ratio between these two values is the closest possible to the given input ratio. If multiple possible combinations of gears yield the same ratio, select the pair with the smallest total number of gears.

Example

Sample Input	**Sample Output**
3	48 16
3.04	76 25
2.414	70 29

This page intentionally left blank.

5 Text Justification

A common task performed by word processors and desktop publishing programs is justifying a paragraph of text, breaking lines at appropriate places given a fixed column width. In this problem, you will write a program to perform simple text justification.

Input to your program will consist of pairs of lines. The first line in each pair is an integer between 0 and 100. If this number is 0, the program has reached the end of input and should terminate. Otherwise, the number indicates the maximum number of characters on each justified line output by your program. The second line is a paragraph to be formatted; this line contains only printable characters (i.e., ASCII 32 through 126), and is guaranteed to be no longer than 2000 characters. In addition, this line is guaranteed to contain at least one character that is not a space. Your program should then format the paragraph according to the rules described below, then write out the resulting formatted lines, followed by a blank line, to standard output.

Each paragraph should be formatted as follows:

- Each formatted line should not contain any spaces at the beginning or at the end, and should be no longer than the maximum width specified as the first line in the pair of lines in the input.
- Line breaks should occur only at spaces in the original input paragraph. If a "word" will fit on a previous line, it should not be placed in the following line. In other words, each line should be filled up before the next line is started. Here, a "word" is defined as a maximal contiguous sequence of characters that are not space. The length of each word in the input is guaranteed to not exceed the given maximum width.
- All spaces at the beginning of or at the end of the paragraph should be ignored. The number of spaces between each pair of adjacent words in the input paragraph should be preserved in the output, except at line breaks.

Again, please be reminded that exactly one blank line must be output after each formatted paragraph to signify that the program is done with the paragraph.

Sample Input

```
10
Please make check payable to Harvard Univ.
20
Take the T to Kendall.  Walk across the street.
0
```

Sample Output

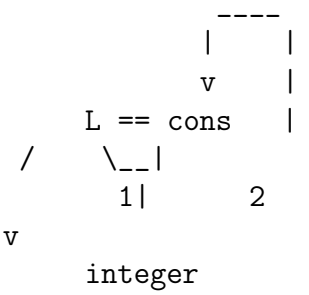
```
Please
make check
payable to
Harvard
Univ.

Take the T to
Kendall.  Walk
across the street.
```

Note that there is one blank line at the end of the sample output above.

6 Cyclic Graph Unification

Preface

Labeled trees are commonly used to represent data types. Cyclic labeled graphs can also be so used, as in:

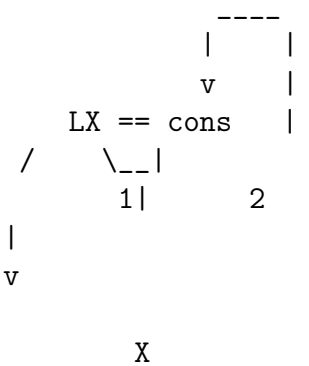


which represents an infinite list of integers. Here 'cons' denotes the type of a data structure with two components: the first component is the first item on the list, and the second component is the rest of the list. L is a variable whose value is the 'cons' node; L can be used to denote the data type represented by this node.

In such a graph, the nodes have labels that are identifiers, and the arrows originating at a node have labels 1, 2, 3, ... that are consecutive integers beginning with 1. The targets of these labels are called the 'children' of the node, even though this is a graph and not a tree, and there can be cycles (so there is no good notion of descendant or ancestor).

Each node represents a data type. We will use variables whose values are nodes of such a graph. We will used the convention that the labels of nodes will begin with a lower case letter, and the names of variables will begin with an upper case letter.

A data type can also have unknown parts that are denoted by variables which represent graph nodes. For example, the graph:



can represent a type in which X is some yet unknown graph node. Here X has been used as the label of a graph node that has NO children: we call this a 'variable node'. So a graph with variable nodes represents a data type with unknown pieces.

Compilers need to determine when two types are the same. We do this with the help of a 'graph equivalence relation'.

An equivalence relation on graph nodes is a set of subsets of the nodes such that each node belongs to exactly one subset. The subsets are called 'equivalence classes'. Two nodes are said to be 'equivalent' if and only if they belong to the same equivalence class.

A graph equivalence relation is an equivalence relation on graph nodes such that whenever two nodes are equivalent then:

Either

1. One of the nodes is a variable node (i.e. has a label beginning with a capital letter.)
2. Both the following are true:
   (a) Both nodes have the same label and number of children.
   (b) Corresponding children of the nodes are equivalent.

Given two nodes, computing the smallest graph equivalence relation that makes the two nodes equivalent is called 'unifying' the nodes. Two nodes representing types are said to represent the same type if and only if the nodes can be unified.

Most pairs of graph nodes, of course, CANNOT be unified. That is, the attempt to compute the smallest graph equivalence relation making them equivalent will fail.

Problem

You are given a graph and are asked to compute the smallest graph equivalence relation that unifies two nodes in the graph, or to indicate that there is no such relation.

The nodes of the graph are numbered with consecutive integers beginning with 0. The label of each node is a single letter. A node is a variable node if and only if its label is a capital letter.

An arrow from a node to one of its children is represented by the node number of the child. Variable nodes have no children.

There can be at most 100 nodes and each node can have at most 10 children.

You are to read in a graph and a pair of node numbers, and try to unify the given pair of nodes. The output of a successful unification is an equivalence relation, and for each node you are asked to print the the name of the equivalence class to which the node belongs. Here the name of the equivalence class, which is a subset of nodes, is defined to be the smallest node number of any node in the equivalence class.

We suggest the following algorithm. Start with a set of equivalence classes each containing a single node. You will discover when two such classes need to be merged. First the classes containing the two nodes to be unified will need to be merged.

When you discover that two classes need to be merged, check (2a), and if it is violated, fail. Otherwise merge the classes, and then use (2b) to find more classes that need to be merged. If there are no more, you are done.

Input

The input consists of one or more runs. Each run consists of one or more node definitions, followed by a '#' character, followed by two node numbers.

A node definition is a line in the format:

```
<node-number>: <node-label> <child>*;
```

The <node-number>s are consecutively increasing integers beginning with 0: the first line of a run has node number 0, the second node number 1, etc.

The <node-label> is a letter.

There can be 0 to 10 <child>ren, each of which is a node number that is a non-negative integer.

The input can contain whitespace, SPACEs, TABs, or LINE FEEDs, before or after any node number, node label, or punctuation mark (':', ':', or '#'). Or such whitespace may be omitted, except between consecutive node numbers. Input ends with an end of file.

Output

For each run in which the unification is NOT successful, the output consists of the two lines:

1. A blank line.
2. A line containing 'NOT UNIFIED'.

For each run in which the unification IS successful, the output consists of the lines:

1. A blank line.
2. A line containing 'UNIFIES'.
3. For each node in order of node number, a line of the format:

```
<node-number>==<class name>
```

where the class name is the smallest node number of any node in the equivalence class of the node whose number is on the left of the '=='.

No output line is to have SPACE or TAB characters.

The output must be PRECISELY as described.

Sample Input

```
0: c 1 0;
1: i;

2: c 3 2;
3: X;
# 0 2

0: c 1 0;
1: i;

2: c 3 2;
3: j;
# 0 2

0: a 1;
1: b 2 0;
2: X;

3: a 4;
4: b 5 7;
5: f 6;
6: Y;
7: a 8;
8: b 9 3;
9: f 10;
10: g;
# 0 3
```

Sample Output

```
UNIFIES
0==0
1==1
2==0
3==1

NOT UNIFIED

UNIFIES
0==0
1==1
2==2
3==0
4==1
5==2
6==6
7==0
8==1
9==2
10==6
```

7 Matching Warriors

The Kingdom of Irfrissable Polytudinous Warriors (KIPW) has a tournament every year between its two tribes, the Kay (K), and the Kaykay (KK). This consists of contests between individual warriors, each tribe having the same number, and the tribe with the most winning warriors wins the coveted thud cup.

Its up to the tribe that lost last year (currently the K) to choose which warriors will fight each other. The K are using science to do this, and have derived the following method of determining the probability that warrior i will defeat warrior j. To this end the K have assigned scores for 6 skills to each warrior, so the scores for warrior i are $s[i][k]$ for $k =$ 0,1,2,3,4,5. Then the probability of warrior i defeating warrior j is $I/(I+J)$, where

$$\begin{aligned} I &= max\{I', 0\} \\ I' &= max\{s[i][k] - s[j][k] : k = 0, 1, 2, 3, 4, 5\} \\ J &= max\{J', 0\} \\ J' &= max\{s[j][k] - s[i][k] : k = 0, 1, 2, 3, 4, 5\} \end{aligned}$$

and in the special case that $I = J = 0$, the probability is 0.5.

It is your task to compute for K the matching of warriors that will give K the greatest expected number of victories.

Input

For each of several data sets:

Line 1 The number n of warriors in each tribe, $0 < n \leq 20$

Lines $2..1+n$ One line for each warrior of K, where the warriors are numbered 1 .. n in increasing order, and the line for warrior i contains:
$s[i][0]s[i][1]...s[i][5]$,
where $0 \leq s[i][k] \leq 10$ for every i and k. The $s[i][k]$ are all integers.

Lines $2+n..1+2n$: Ditto for the warriors of KK.

The input ends with a line containing a single 0.

Output

One line per instance containing 'Instance #:' followed by the KK warrior numbers matched to the warriors of tribe K in increasing order of the K warrior numbers. E.g., '2 3 1' means K warrior 1 fights KK warrior 2, K warrior 2 fights KK 3, K 3 fights KK 1.

Sample Input

```
2
1 0 0 0 0 0
0 2 0 0 0 0
0 0 1 0 0 0
0 0 0 3 0 0
3
1 0 0 0 0 0
0 2 0 0 0 0
0 0 3 0 0 0
0 0 2 0 0 0
0 1 0 0 0 0
3 0 0 0 0 0
0
```

Sample Output

```
Instance 1: 2 1
Instance 2: 3 2 1
```