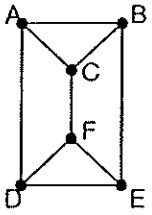
With corrections

Circuit Tracing

Problem A

The simplest non-trivial junction of wires is one which connects three wires at one point. Quite complex circuits can be created from these three-wire junctions. Below we show a fairly simple example, together with a description of it made by alphabetically listing, for each junction, the other junctions it is connected to.



A:	B	C	D		D:	A	E	F
B:	A	C	E		E:	B	D	F
C:	A	B	F		F:	C	D	E

There is a much more compact way of writing these descriptions using the facts that every junction has precisely three neighbours (so the rows need not be labelled), and if A is connected to B, then B is connected to A (so B's list need not contain A, and so on). Also, since the letters A, B,... are arbitrary, every circuit can be lettered so that A is connected to B, C, and D—hence the first row is not needed at all. All that is really needed is:

C E F E F F

From this we can regenerate the original table (here the omitted entries which have been regenerated are shown underlined):

A:	B	C	D		D:	A	E	F
B:	A	C	E		E:	B	D	F
C:	A	B	F		F:	C	D	E

Write a program which reads circuit descriptions in the compact form and outputs the full table for each circuit.

Input is from a file named `PROBLEMA.DAT` and contains several circuit descriptions in compact form. Each description consists of a sequence of capital letters separated by spaces and/or newlines. The descriptions are not explicitly separated because the process of expansion determines the end of each description. The file is terminated by the character #.

Output for the nth circuit consists of a line reading "`Circuit #`n`:`" followed by one line for each junction, in alphabetical order. Each line should be of the form "w`:` x y z", where the neighbours x, y, z are in alphabetical order and each preceded by a single space. There should be no blank lines anywhere in the output.

Sample input

```
C D D C
E F E F
F #
```

Sample output

```
Circuit #1:
A: B C D
B: A C D
C: A B D
D: A B C
Circuit #2:
A: B C D
B: A C E
C: A B F
D: A E F
E: B D F
F: C D E
```

Tele-Typos

Problem B

In the Good Old Days, when computers were rooms and required air conditioning, teletypes really were typewriters at a distance from the main computer. Since characters really were typed onto paper as the keys on such a teletype were pressed, there was no way to erase the display in the event of typing mistakes. One popular way of showing that the delete key had been pressed was to reprint the character that was thus deleted. For example, "LISRRT" appearing on the printout would probably have arisen from the keystrokes L, I, S, R, *delete*, T and have been sent to the computer as "LIST".

Write a program which, given a dictionary of correctly spelled words and command lines from such a printout, deletes all double letters that are due to typos and outputs the command lines in the form in which they are sent to the main computer.

Input is from a file named `PROBLEMB.DAT`, and consists of two types of lines, terminated by a line containing only #. Both types of line are at most 80 characters in length. Lines starting with < and ending with > are command lines to be corrected. Any other lines contain space-separated words to be added to the dictionary. A dictionary word can be up to 20 letters in length, and the dictionary can contain up to 500 words. No word appears more than once in these dictionary lines.

Apart from the <, >, and # characters mentioned, the only characters that appear in the input are spaces and uppercase letters.

For each command line input, your program should output a line in exactly the same format as the input line, with each misspelled word replaced by a word in the dictionary. The dictionary consists of those words which have been added by dictionary lines before the command line being processed.

It is possible that several different dictionary words may be viable corrections for a word in an input command line. (For example, consider a dictionary containing the words BE and BELL with the input BELEEL.) Your program should choose the word which requires the minimum number of corrections. (Here BELL requires only 1, compared to BE's 2.) However, occasionally several dictionary words may all be formed with the same minimal number of corrections. In this case, your program should choose the first word in alphabetical order.

It is also possible that, for some input word, *no* combination of typo removals can find a corresponding dictionary word. This may indicate a deficiency in the dictionary or that what appear to be spaces delimiting the input word may in fact be typos. In this case, your program should instead output a line containing the message "`Not in dictionary or spaces in error`".

Sample input

```
THE BAN LIST
<BABBLLOON>
BALLOON  BABBOON  SIC
<BABBLLOON LISRRT   >
<BAL  LOON>
<  BAMMN  THE  LISRSSRT  >
#
```

Sample output

```
<BAN>
<BABBOON LIST   >
Not in dictionary or spaces in error
<  BAN  THE  LIST  >
```

Radio Broadcasting Systems

Problem C

You run a small telecommunications company specialising in setting up private radio broadcasting systems. These systems have one transmitter and many receivers. To keep the system as cheap as possible, the transmitter must be centrally located and use the lowest possible power. Since the transmitter radiates power uniformly, the optimum system design is to place the transmitter at the centre of the smallest circle that encompasses all of the receivers. Also, because of the accuracy of the measurements used, all coordinates, including those of the transmitter, are integers.

The input comes from a file named `PROBLEMC.DAT`, and describes a number of broadcasting system layouts. Each begins with a single line containing the number of receivers, which may range from one to fifty. Following this is one line for each receiver giving that receiver's (x, y) coordinates, where x and y are integers between 0 and 5000. The end of the input is denoted by a system layout containing 0 receivers.

For each system layout print, on a single line separated by single spaces, the (x, y) coordinates of the centre of the smallest circle enclosing all of the receivers, and the radius of that circle. The centre's coordinates must, of course, be integral, and the radius must be given rounded to two decimal places. If the smallest circle can be placed in a number of possible positions, choose the position with the smallest x coordinate. If there is still ~~a number of~~ possibilities choose the position with the smallest y coordinate.

more than one

Sample input

```
2
10 11
11 20
3
20 100
50 50
100 100
0
```

Sample output

```
10 16 5.00
60 90 41.23
```

Depot Placement

Problem D

Your company runs a food delivery service in a large state. Your customers own shops in towns all over the state. You are planning to build a large central depot from which to supply all of your customers, and this will also need to be built in one of the towns. You need to place the depot such that the total delivery time to all towns is minimised. Your delivery trucks take one full load of food to a shop and then return to the depot. The delivery time is the time taken to get from the depot to the town and back again assuming that the driver takes the fastest possible route, and takes precisely 30 minutes to unload the truck at the destination. The roads connecting the towns are quite different in quality, and for some roads it takes longer to travel in one direction than the other. There are even a number of one-way roads. To simplify all of this, you have established for each town a list of all the roads that lead out to other towns, and how long it takes to get to each of these other towns.

Input comes from a file named `PROBLEMD.DAT`, and contains data for a number of states. Each state description starts with a single line containing the number of towns in the state. Following this is one line per town. Each of these lines contains the number of roads leaving the town, followed by pairs of numbers containing the number of the town that the road ends at, and how long it takes to get there (in minutes). The towns are numbered from 1 up to the total number of towns (which is between 2 and 50).

The maximum number of minutes for any road segment is 120 and the minimum is 10; the minimum number of roads leaving any town is 1 and the maximum number is one less than the number of towns in the state. There will be no more than one road directly connecting any two towns. The input file will be terminated by a state description containing 0 towns.

For each state you must output a single line giving the number of the town from which the total delivery time is minimum, and what that total delivery time is. If there are multiple towns with the minimum, you must output the town numbers in ascending order with the total delivery time at the end. All numbers output are to be separated by single spaces.

Sample input

```
5
2 2 70 4 55
3 1 70 3 30 5 65
3 2 30 5 40 4 35
3 1 55 3 35 5 50
3 2 65 3 40 4 50
2
1 2 35
1 1 25
0
```

Sample output

```
3 540
1 2 120
```

Isotope Ratios

Problem E

The nucleus of an atom consists of protons and neutrons. The number of protons is called the atomic number and determines the chemical properties of the atom. An element is a substance composed of atoms which all have the same number of protons. Each of the 100 odd elements is given an identifying symbol like O for oxygen, H for hydrogen and Cl for Chlorine.

The atoms which compose an element do not all have the same number of neutrons, but the number of neutrons in each atom of an element does not vary by much (10 or less) because atoms with too many or too few neutrons are unstable (radioactive). The isotopes of an element are the different types of atom which can occur—all the types have the same number of protons but different numbers of neutrons. The atomic weight of an atom is the sum of the number of protons and neutrons; so isotopes are characterised by their atomic weights. Usually one isotope is very much more common than all others (some elements, e.g., Fluorine, in fact have only one stable isotope) but some have a more even distribution: for example, Chlorine has about 76% of the naturally occurring element with atomic weight 35 and the remaining 24% with atomic weight 37.

The situation becomes more complicated when we consider molecules, which are combinations of atoms; for example, a molecule of water contains two hydrogen atoms and one oxygen atom (H_2O). The molecular weight of a molecule is the sum of the atomic weights of the component atoms. Since each element present may have a range of atomic weights, the range of molecular weights for different molecules of the same compound may be large. Consider the following:

The elements Chromium (symbol Cr) and Chlorine (Cl) have the following sets of isotopes:

Atomic weight	Abundance
50	4.35%
52	83.79%
53	9.50%
54	2.36%

Atomic weight	Abundance
35	75.77%
37	24.23%

Thus the molecular weight of a molecule of Chromium Chloride ($CrCl_2$) can range from 120 (50 + 35 + 35) to 128 (54 + 37 + 37). In fact the possible molecular weights and their relative abundances work out to:

120	2.50%
122	49.70%
123	5.45%

124	32.38%
125	3.49%
126	5.79%

127	0.56%
128	0.14%

It is easy to compute that the mean molecular weight is 123.02, while the median molecular weight is 124.5. You are to write a program that will read in a chemical formula and determine the mean molecular weight and the median molecular weight. If there are an odd number of possible molecular weights, the median molecular weight is simply the middle one; if there are an even number, as above, the median is halfway between the middle two.

Input is from a file called `PROBLEME.DAT` and consists of two parts. The first part consists of a series of lines, one for each element being considered. Each of these lines consists of the symbol for an element (a one or two letter string with the first (or only) letter in uppercase), followed by a list of atomic weights (integers in the range 1 to 300) and percentage abundances (real numbers with two digits after the decimal place). Each line is terminated by a single 0. The abundances always sum to 100.00. This part of the file is terminated by a line consisting of a single #.

The second part of the file consists of a series of molecular formulae for which the mean and median molecular weights are desired. These are written in conventional notation which attempts to convey something of the structure of the molecule as well as its composition. Thus a molecule of aspirin is normally written as $CH_3COOC_6H_4CO_2H$. In the file such a molecule would be written as `CH3COOC6H4CO2H`. In addition, certain subgroups can appear more than once in a molecule in which case they are bracketed: for instance Ammonium Sulphide (conventionally written as $(NH_4)_2S$) would appear as `(NH4)2S` in the file. This part of the file will also be terminated by a line consisting of a single #. All substances given as molecular formulae have fewer than 7 different elements and no more than 50 atoms in each molecule.

Output will consist of a line for each molecular formula in the input file, giving the average molecular weight correct to two decimal places, and the mean molecular weight as an integer or an integer followed by .5. The two numbers are separated by a single space, and there are no spaces in front of the first number.

Sample input

```
Cr 50 4.35 52 83.79 53 9.50 54 2.36 0
Cl 35 75.77 37 24.23 0
F 19 100.00 0
C 12 98.89 13 1.11 0
H 1 99.99 2 0.01 0
O 16 99.76 17 0.04 18 0.20 0
N 14 99.64 15 0.36 0
S 32 95.02 33 0.75 43 4.22 36 0.01 0
#
CrCl2
CH3COOC6H4CO2H
(NH4)2S
F20
#
```

Sample output

```
123.02 124.5
180.12 192.5
68.48 78.5
380.00 380
```

Uniform Marks

Problem F

The Not-So-Advanced School of Statistics has decided that it wishes all of its exam results to have a uniform distribution. They also want to round people's marks to multiples of ten from 10 to 100. You are to write a computer program to do this for them. This means that the number of students who end up getting a mark of 10 should be the same as the number who get 20, or 30, and so on. If it is not possible to distribute the students evenly, then give them higher rather than lower marks. The difference between the numbers of students in any two grade ranges should never be more than 1. The final marks should clearly order the students in the same way as their pre-normalised marks. If two students have the same original mark, then they are to be sorted by student number, with lower student numbers being rated more highly.

Input comes from a file named `PROBLEMF.DAT`, and contains results for a number of different exams. For each exam there is a line containing the number of students. Following this is one line for each student containing their student number followed by their mark. The student numbers are in the range 95000 to 95999, and the marks are in the range 0 to 100. There are from 1 to 100 students in each exam. The input file is terminated by an exam containing 0 students.

For each exam, you are to output the number of students in that exam on a single line followed by one line for each student, ordered as described above. These lines should be formatted as follows: the student number right justified in a field of 5 digits, a single space, and then the mark right justified in a field of 3 digits.

Sample input

```
6
95001  32
95002  45
95003  55
95004  99
95005   0
95006 100
12
95100  35
95101  35
95102  68
95103  35
95104  68
95110  77
95111  99
95112   0
95120  35
95121  68
95122  99
95123 100
0
```

Sample output

```
6
95005  50
95001  60
95002  70
95003  80
95004  90
95006 100
12
95112  10
95120  20
95103  30
95104  40
95100  50
95121  60
95110  70
95122  80
95111  90
95123 100
```

The Primes of Phibes

Problem G

Dr Phibes of Hyperbolic Fabrications Ltd is studying the following function F:

$$F(a,n) = \text{the remainder when } a^{n-1} \text{ is divided by } n = a^{n-1} \bmod n$$

He knows that if he can find a number $1 < a < n$ such that $F(a,n) \neq 1$ then n cannot be prime. For example, $F(2,9) = 2^8 \bmod 9 = 4 \neq 1$, and 9 is indeed not prime.

He is convinced that he can prove Fermat's Theorem (and thus gain worldwide adulation) by studying the pattern of values of a for various ranges of non-prime n.

Unfortunately, a list of prime numbers has become mixed up with his collection of interesting (a,n) pairs, and the a values have been lost altogether. Luckily, he is so familiar with his n values that he has been able to separate the mess of numbers into groups of five, each of which he is sure contains exactly one prime number.

Write a program that will take each group of five numbers in turn, identify the prime, and produce an a value for each of the other four numbers. Dr Phibes has given you one additional piece of information that he thinks will be very valuable:

$$(x \cdot y) \bmod n = (x \bmod n) \cdot (y \bmod n) \bmod n$$

Input is from a file named `PROBLEMG.DAT` and consists of a series of lines, each line containing five positive integers all less than 2^{30}, separated by spaces. This file is terminated by a line containing five zeroes.

For each non-zero line in the input, output a line containing one item for each number, separating these items with a single space. For the single prime on each line, this item should be the word "PRIME". For the other four numbers, n, the item is the smallest $a > 1$ such that $F(a,n) \neq 1$.

Sample input

```
14 15 16 17 18
341 277 1387 29341 1105
0 0 0 0 0
```

Sample output

```
2 2 2 PRIME 2
3 PRIME 3 13 5
```

Database Fields

Problem H

Common databases store numbers as strings of digits. If a database field is described as "Number, width n, decimal places d", then n character positions are allocated for it by the database, one of which is used for the decimal point. For example, if $n = 6$, $d = 2$, then the maximum number which can be stored is 999.99. Typically the maximum value of n is 20 and the maximum value of d is $n - 2$, to allow for an initial "0." (so $n \geq 2$ always).

You work for a firm which is designing a database system which will only store positive numbers (your boss doesn't believe in negatives). You can see that there is a much more efficient way of storing numbers: since d, the number of decimal positions, is stored in the database header, the decimal point can be omitted from each number, and the number then stored as a binary integer. For example, in the $n = 6$, $d = 2$ case, integers up to 99999 need to be stored, which can be done in 17 bits, i.e., three bytes instead of six.

To convince your boss that using your system will not adversely affect database calculations, you need to compute the maximum number (rounded up*) which can be stored in a field under the old system. Your boss will be most amenable if you write this number in words, using the old English notation:

one million = 10^6 one billion = 10^{12} one trillion = 10^{18}

Input comes from a file named `PROBLEMH.DAT`, and consists of lines each containing two integers, which are values for n and d respectively. Input is terminated by a line containing two zeros.

For each non-zero line in the input, output one line containing n, d, the size in bytes of a field stored in the old system, the size in bytes of a field stored in your system, and the old English language form of the maximum number (rounded) in the old system. Numbers should be right justified in a column of width 2, with a single space placed between items, so that the output is tabulated as shown.

* to the next highest power of 10

Sample input

```
2 0
6 2
18 3
0 0
```

Sample output

```
 2  0  2  1 ten
 6  2  6  3 one thousand
18  3 18  8 one hundred billion
```