University of Scranton
ACM Student Chapter / Computing Sciences Department
16th Annual High School Programming Contest (2006)

Problem 1: Abundant Numbers

For any natural number (i.e., nonnegative integer) m, let $f(m)$ be the sum of the positive divisors of m, not including m itself. As examples, $f(28) = 1 + 2 + 4 + 7 + 14 = 28$, $f(12) = 1 + 2 + 3 + 4 + 6 = 16$, and $f(51) = 1 + 3 + 17 = 21$.

If $m = f(m)$, m is said to be *perfect*; if $m < f(m)$, m is said to be *abundant*; if $m > f(m)$, m is said to be *deficient*.

Develop a program that, given a positive integer m, reports whether m is abundant, perfect, or deficient.

Input: The first line contains a positive integer n indicating how many numbers are to be processed. Each of the next n lines contains one such number, which will be a positive integer.

Output: For each number given as input, report on a single line whether it is abundant, perfect, or deficient. See sample output below for exact formatting.

Sample input	Corresponding output
3	12 is abundant
12	28 is perfect
28	51 is deficient
51	

University of Scranton
ACM Student Chapter / Computing Sciences Department
16th Annual High School Programming Contest (2006)

Problem 2: Alphabet Soup

Develop a program that, given a "bowl of alphabet soup" along with one or more "messages" (i.e., character strings), determines, for each message, whether or not the bowl contains all the letters necessary to form the message.

Regarding the case (lower or upper) of letters, your program should treat a as being indistinguishable from A, b as being indistinguishable from B, etc. Hence, an occurrence of, say, F in a message is "covered by" an occurrence of f in the soup.

Note that the number of occurrences of each letter matters: if, for example, there are three occurrences of e/E in the soup but four (or more) occurrences of e/E in a message, that message cannot be formed from the letters in the soup.

As for non-letters appearing in a message (e.g., spaces, digits, punctuation symbols, etc.), they should be ignored in determining whether the message can be formed in the soup.

Input: The first line contains a positive integer n indicating how many instances of the problem are described on subsequent lines. Each instance is described beginning with a line containing a string of letters, in no particular order and (typically) containing duplicates. This line represents the collection of letters in the bowl of soup. You may assume that this string has length between one and 60, inclusive. On the next line is a single positive integer m indicating how many messages are to follow. Each of the next m lines contains a message, which is simply a string of characters, possibly including non-letters (such as spaces and punctuation marks). You may assume that each message has length between one and 60, too. The line following the m-th message is blank.

Output: For each instance of the problem described in the input, generate a line of output that echos the string representing the bowl of soup. On subsequent lines, echo each given message and report (using either "YES" or "NO") whether or not it can be formed using the letters in the soup. (See sample output below for exact formatting.) After the last message, print a blank line.

Sample input and output is on the next page.

```
Sample input
------------
2
lhLoe
3
HELLO
eel
e L,$H.

ToBeOrNotToBeHamletSaid
3
"To be, or not to be", Hamlet said.
I'm beat!
No horses ebb
```

```
Corresponding output
--------------------
lhLoe
HELLO : YES
eel : NO
e L,$H. : YES

ToBeOrNotToBeHamletSaid
"To be, or not to be", Hamlet said. : YES
I'm beat! : YES
No horses ebb : NO
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
16th Annual High School Programming Contest (2006)

Problem 3: Grade Book

In a *gradebook*, a teacher records the name of a course, the names of the students taking that course, the *scores* that the students have earned on *scored works* (e.g., tests, quizzes, homework assignments, etc.), and the *weights* associated with those scored works. Here we shall assume that all scores are integers in the range zero to 100. Regarding the concept of a weight, a particular test might be worth 15% of the course grade, whereas a particular homework might be worth only 6%, for example. We refer to the values 15% (or, equivalently, 0.15) and 6% (or 0.06) as the weights of the test and homework, respectively.

Suppose that k scores have been recorded for each student and that the weights associated with the scored works are $w_1, w_2, \ldots, w_k$. If a student's scores on those works are $s_1, s_2, \ldots, s_k$, then her *weighted average* is

$$(s_1 \cdot w_1) + (s_2 \cdot w_2) + \cdots + (s_k \cdot w_k)$$

While a course is still in progress, one would expect the recorded weights to add up to something less than 100% (or, equivalently, 1). (For that matter, sometimes even at the end of a course the weights don't add up to exactly 100%.) Hence, it makes sense to *normalize* a weighted average by multiplying it by $\frac{1}{W}$ (or, equivalently, dividing it by W), where W is the sum of the weights.

Develop a program that, given a teacher's gradebook, calculates each student's (weighted and normalized) course average and course grade. A course average in the interval $[0, 60)$ yields a course grade of **F**. The interval $[60, 70)$ yields **D**, $[70, 80)$ yields **C**, $[80, 90)$ yields **B**, and $[90, 100]$ yields **A**.

Input: The first line contains a positive integer n indicating how many gradebooks are to be processed. Each gradebook is described by a line containing the name of the course, a line containing the number m of students in the course and the number k of scores that have been recorded for each student, a line containing the respective weights of those k scores, each a real number with one digit before the decimal point and two digts after, and then m lines containing student names and scores, one student per line. A student's name occupies the first nine positions on a line and is followed by a space, which is followed by the student's k scores, separated by spaces. Each score is an integer between 0 and 100. The last line is blank.

Output: For each gradebook given as input, the program should produce $m+2$ lines of output, where m is the number of students in the course. The first line should contain the name of the course. Each of the following m lines should contain a student's name, followed by her weighted-and-normalized average, followed by her letter grade. Students should appear in the output in the same order that they appeared in the input. The last line should be blank.

Sample input:

```
-------------
2
Geometry
4 5
0.10 0.20 0.24 0.16 0.20
Alice      78  90  95  82  80
Bill       50  60  70  80  90
Carol     100 100 100 100 100
Dave       80  75  64  90  80

Abstract Algebra
2 2
0.60 0.50
Mort       80  80
Zelda      90  70
```

Corresponding output:

```
---------------------
Geometry
Alice      86.36 B
Bill       71.78 C
Carol     100.00 A
Dave       76.40 C

Abstract Algebra
Mort       80.00 B
Zelda      80.90 B
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
16th Annual High School Programming Contest (2006)

Problem 4: Paths in Trees

In mathematics, a *tree* is a structure that ... well ... resembles those large plants that grow in forests, orchards, and city parks. Strangely, however, the trees found in math books are usually drawn upside down compared to the trees found in your back yard. Consider the figure below, for example. In it is depicted a tree. The numbers are used simply to provide names for the *nodes*. In a tree, pairs of nodes are connected by *edges*. Specifically, every node in a tree, except for one that we call the *root*, is connected by an edge to a node above it, which we refer to as its *parent* node. In the figure, node 8 is the root and each of nodes 3, 5, and 12 have node 8 as its parent. Meanwhile, node 7's parent is node 3.

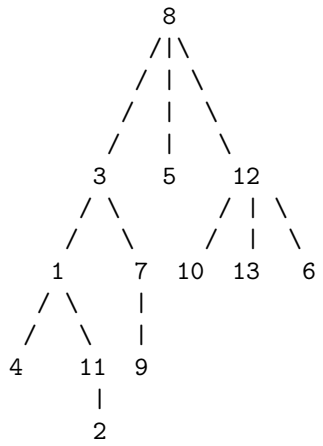

Figure 1: An example tree

A *path* in a tree is a sequence of nodes $\langle x_1, x_2, \ldots, x_k \rangle$ such that no node appears more than once and such that, for each i satisfying $1 \leq i < k$, x_i and x_{i+1} are connected by an edge. One of the interesting properties of a tree is that, for every pair of nodes, there is a unique path that connects them. For example, $\langle 11, 1, 3, 7 \rangle$ is the unique path from node 11 to node 7.

Develop a program that, given as input a tree and a list of pairs of nodes in that tree, reports for each given pair the unique path that connects them.

Hint 1: For this problem, an effective approach for representing a tree is to maintain an array `parentOf[ ]` such that, if k is the parent of j, then `parentOf`$[j] = k$.

Hint 2: One of the keys for finding the path connecting two nodes in a tree is to find their *nearest common ancestor*. The path from node 11 to node 7, for example, goes from 11 up to their nearest common ancestor, node 3, and then down from there to node 7. Every path in a tree has this form. (In some cases, however, the upward portion or downward portion of the path has no edges.)

Input: The first line contains a positive integer n indicating how many instances of the problem are to be solved. Each instance is given by a tree and a list of node pairs. The tree is described on two lines, the first of which contains a positive integer m indicating how many nodes are in the tree. (The nodes are assumed to be named by the integers 1 through m.) The second line contains a sequence of m integers identifying each node's parent. Specifically, the first number identifies node 1's parent, the second number identifies node 2's parent, etc. The root node has no parent; hence we put zero in that position. (The first tree described in the sample input below is the one depicted in the figure above.) The node pairs are given beginning on the third line, which contains a positive integer r indicating how many such pairs are to follow. On each of the next r lines is a pair of integers in the range $1..m$ identifying two nodes in the tree.

Output: For each tree given as input, and each pair of nodes given with respect to that tree, report the unique path from one to the other, as shown in the sample output below. Use a blank line to separate the output associated with one tree from that associated to the next.

```
Sample input                          Corresponding output
------------                          --------------------
2                                     path from 11 to 7: 11 1 3 7
13                                    path from 2 to 3: 2 11 1 3
3 11 8 1 8 12 3 0 7 12 1 8 12         path from 12 to 12: 12
5                                     path from 13 to 4: 13 12 8 3 1 4
11 7                                  path from 8 to 6: 8 12 6
2 3
12 12                                 path from 2 to 3: 2 1 3
13 4
8 6
3
0 1 1
1
2 3
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
16th Annual High School Programming Contest (2006)

Problem 5: Binary Operation Expression Evaluation

To say that $\oplus$ is a binary operation over the set S is to say that $x \oplus y \in S$ for every $x, y \in S$.

If S is finite, we can describe any binary operation over S by means of a table. Consider, for example, the following, which describes a binary operation $\oplus$ over the set $\{a, b, c\}$.

```
    a  b  c
  +---------+
a | b  a  c |
b | b  c  b |
c | a  c  b |
  +---------+
```

To find the value of $a \oplus c$, for example, we look in the row labeled a and the column labeled c. There we find c, which is to say that $a \oplus c = c$. As another example, $c \oplus c = b$.

As an example of an expression involving multiple applications of $\oplus$, consider

$$b \oplus a \oplus (b \oplus (a \oplus a \oplus c) \oplus (a \oplus b)) \oplus c$$

When it is understood that $\oplus$ is the only operation involved, we can omit references to it when writing an expression. Doing so, the above expression becomes

$$ba(b(aac)(ab))c$$

Notice that this expression is ambiguous insofar as it is not clear whether we should interpret, for example, the subexpression aac as $(aa)c$ or as $a(ac)$. Let us agree to resolve this ambiguity by assuming that $\oplus$ *associates to the left.* That is, we shall interpret an expression of the form xyz as $(xy)z$. (We follow a similar convention with respect to subtraction by interpreting the ambiguous $x - y - z$ as $(x - y) - z$.)

What follows is a step-by-step evaluation of the expression above. One might call it a *leftmost* evaluation, because, at each step, the leftmost evaluable subexpression (which is underlined) is simplified.

$$\begin{aligned}
\underline{ba}(b(aac)(ab))c &= b(b(\underline{aa}c)(ab))c && (ba = b)\\
&= b(b(\underline{bc})(ab))c && (aa = b)\\
&= b(\underline{bb}(ab))c && (bc = b)\\
&= b(c(\underline{ab}))c && (bb = c)\\
&= b(\underline{ca})c && (ab = a)\\
&= \underline{ba}c && (ca = a)\\
&= \underline{bc} && (ba = b)\\
&= b && (bc = b)
\end{aligned}$$

Develop a program that, given a tabular description of a binary operation $\oplus$ and a sequence of $\oplus$-expressions, evaluates each expression and outputs the result. You may assume that S is comprised of the letters in some initial segment of the sequence $\langle a, b, c, \cdots, z\rangle$.

Input: The first line contains a positive integer n indicating how many instances of the problem will be described on subsequent lines. Each such instance begins with a positive integer m, $1 \leq m \leq 26$, appearing by itself on a line and indicating the size of S. Occupying the next m lines is an $m \times m$ table defining the operation $\oplus$.

For example (assuming that $m \geq 7$), the value of $g \oplus c$ is found in the third column (because c is the third letter in the alphabet) of the seventh row (because g is the seventh letter).

On the line following the table is a positive integer p indicating the number of expressions to evaluate. On each of the following p lines is an expression to be evaluated. You may assume that each expression is syntactically valid and contains no redundant parentheses.

Output: For each expression given as input, display it, followed by a space, an equal sign, another space, and, finally, its simplified form. Output a blank line as the last line of output for each instance of the problem.

```
Sample input
------------
2
3
bac
bcb
acb
2
ba(b(aac)(ab))c
ccb(a(ab))ba
5
abcde
edcba
babab
aaced
ddacb
2
d
a(bed)(cabde)
```

```
Corresponding output
--------------------
ba(b(aac)(ab))c = b
ccb(a(ab))ba = a

d = d
a(bed)(cabde) = a
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
16th Annual High School Programming Contest (2006)

Problem 6: Egyptian Fractions

A *unit fraction* is a fraction of the form $\frac{1}{r}$, where r is a (non-zero) integer. An *Egyption fraction* is a sum of distinct positive unit fractions, so called because this is the manner in which ancient Egyptians expressed fractions in general. For example, they would have written $\frac{3}{5}$ as $\frac{1}{2} + \frac{1}{10}$ (except that they would have used hieroglyphics rather than Arabic numerals).

Develop a program that, given two positive integers x and y, calculates an Egyptian fraction (i.e., a sum of unit fractions) equal to $\frac{x}{y}$. To simplify matters a little, you may assume that $x < 2y$ (i.e., $\frac{x}{y} < 2$).

Input: The first line contains a positive integer n indicating how many instances of the problem are subsequently described. Each such instance is described on a single line containing two positive integers x and y, separated by a space and representing the fraction $\frac{x}{y}$.

Output: For each fraction $\frac{x}{y}$ given as input, your program should generate a single line of output containing the given fraction, a colon, and a list of the denominators, in ascending order, in an equivalent Egyptian fraction. As examples, the sample input and output below indicate that $\frac{3}{5} = \frac{1}{2} + \frac{1}{10}$, $\frac{5}{3} = \frac{1}{1} + \frac{1}{3} + \frac{1}{4} + \frac{1}{12}$, and $\frac{15}{33} = \frac{1}{3} + \frac{1}{9} + \frac{1}{99}$.

Sample input	Corresponding output
3	3/5 : 2 10
3 5	5/3 : 1 3 4 12
5 3	15/33 : 3 9 99
15 33	