University of Scranton
ACM Student Chapter / Computing Sciences Department
11th Annual High School Programming Contest (2000)

Problem 1: Continued Fractions

Recall that a rational number is a ratio of two integers. Typically, we express a rational number in the form n/d (i.e., as a fraction), where n and d are integers, with $d \neq 0$. We refer to n and d as the *numerator* and *denominator*, respectively.

Interestingly, every positive rational number (by which we mean a rational number greater than zero) can be expressed in the form

$$q + \frac{1}{E}$$

where q is a nonnegative integer and E is either the number 1 or an expression of the same form as above. We call such an expression a *continued fraction*.

For example, the rational number $\frac{239}{51}$ is equal to the continued fraction

$$4 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{5 + \cfrac{1}{2 + \cfrac{1}{1}}}}}$$

To describe a continued fraction in a more compact form, we can simply write the sequence of nonnegative integers preceding the plus signs, from top to bottom. For the above example, this yields the sequence $\langle 4\ 1\ 2\ 5\ 2 \rangle$.

For every positive rational number r, there are many (in fact, infinitely many!) different continued fractions equal to r. For example, 239/51 is equal not only to the continued fraction corresponding to the sequence $\langle 4\ 1\ 2\ 5\ 2 \rangle$ (as noted above) but also to the one corresponding to the sequence $\langle 4\ 1\ 1\ 0\ 1\ 5\ 2 \rangle$ (as well as infinitely many others).

Let us restrict attention to continued fractions corresponding to sequences in which zero may not appear, except in the first position. We refer to these as *proper* continued fractions. (For example, the continued fraction corresponding to $\langle 4\ 1\ 2\ 5\ 2 \rangle$ is proper, but the one corresponding to $\langle 4\ 1\ 1\ 0\ 1\ 5\ 2 \rangle$ is not.) It turns out that, for every positive rational number r, there is exactly one proper continued fraction equal to r.

Develop a program that, given a positive rational number as input, produces as output the sequence corresponding to the proper continued fraction to which it is equal.

Hint: Let n and d be positive integers, so that n/d is a positive rational number. Divide d into n, letting q be the quotient and r be the remainder. (That is, let q and r be such that $n = q \cdot d + r$, with $0 \leq r < d$.) If $r = 0$, we have

$$n/d = q = (q-1) + \frac{1}{1}$$

and thus n/d is equal to the continued fraction corresponding to the one-element sequence $\langle q-1 \rangle$. If, on the other hand, $r > 0$, then

$$n/d \;=\; q + \frac{r}{d} \;=\; q + \frac{1}{d/r}$$

and thus the sequence corresponding to the proper continued fraction equal to n/d is the one obtained by inserting q at the beginning of the sequence corresponding to the proper continued fraction equal to d/r.

Input:
The first line of input contains a positive integer m indicating how many rational numbers are to be processed. Each of the next m lines contains a pair of positive integers (separated by a space), the first being the numerator of a rational number and the second being the denominator.

Output:
For each rational number given as input, the output should include the sequence corresponding to the unique proper continued fraction equal to that number. Each such sequence should appear on a separate line, with adjacent elements separated by at least one space.

```
Sample Input:                    Corresponding Output:
-------------                    ---------------------
3                                2 1 2 2
27 10                            3 9 5 6
1029 331                         0 1 13 4 3 1 5 2
4116 4427
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
11th Annual High School Programming Contest (2000)

Problem 2: Robot Rendezvous

Imagine a room in which there are n designated "locations", numbered 1 to n. Two robots, R2 and D2, are placed into the room, possibly at different locations. The robots are programmed so that, whenever a gong is sounded, they simultaneously move to a specified "next" location, according to a given "map". A map indicates, for each location, which location should be visited next. Thus, a map can be viewed as a function f such that, for each integer i satisfying $1 \leq i \leq n$, if a robot is at location i, then, upon hearing a gong, it will move to location $f(i)$.

Develop a program that, given as input the number of locations in the room, the initial locations of R2 and D2, and the (common) map that they have been programmed to follow, determines whether the two robots will ever meet (i.e., be at the same location at the same time), and, if so, the location and time (i.e., after how many gong soundings) of their first meeting. If the robots will *never* meet, that fact should be reported by the program.

Hint: Developing a program that simulates the movements of the two robots is a good approach. However, a program that performs such a simulation, stopping only when the robots meet, is not a correct solution, because it gets trapped in an infinite loop in the case that the robots never meet.

Input

The first line of input contains a positive integer m indicating how many instances of the problem are to be solved. The next $3m$ lines contain the m instances, three lines per instance. The three lines comprising each instance are as follows: The first line contains a positive integer n indicating the number of locations in the room. (You may assume that $n \leq 100$.) The second line contains two positive integers (in the range 1 to n) that indicate the initial locations of R2 and D2, respectively. The third line contains a sequence of n positive integers, each in the range 1 to n, representing the map. For each i satisfying $1 \leq i \leq n$, the i-th number in the sequence indicates the location that is to be visited immediately after location i. (Viewing the map as a function f, the i-th number in the sequence gives the value $f(i)$.)

Output

For each instance given as input, the output should include a line that says either "Robots never meet" or "Robots meet at location p after t moves", where p and t are filled in correctly. In particular, p and t should describe the location and time (i.e., number of moves preceding) the robots' *first* meeting.

Sample Input:	Corresponding output:
3	Robots never meet
10	Robots never meet
3 4	Robots meet at location 5 after 2 moves
5 6 8 7 8 1 9 5 4 6	
10	
3 2	
5 6 8 7 8 1 9 5 4 6	
10	
3 6	
5 6 8 7 8 1 9 5 4 6	

University of Scranton
ACM Student Chapter / Computing Sciences Department
11th Annual High School Programming Contest (2000)

Problem 3: Programming Contest Scoreboard

Imagine a computer programming contest that works as follows. A number of programming problems is posed to each of several competing teams, and each team tries to solve as many of the problems as it can. Each time a team has developed what they believe to be a correct solution to one of the problems, they submit it to the judge. The judge then determines, by subjecting the submitted program to a rigorous testing process, whether or not it is a correct solution. The team is notified of the judge's verdict. In the case that the judge deemed the submission to be incorrect, the team is free to modify their program and to submit it again.

Teams are ranked according to the number of problems solved. In order to break ties among teams that solved the same number of problems, the amount of time that they used in developing their solutions is taken into consideration. Specifically, when a team is given credit for having solved a particular problem, the time charged against the team is equal to the number of minutes that had elapsed (since the contest began) when the correct program was submitted, plus twenty (20) minutes for each prior submission of an incorrect program for that problem. For example, if a team submits a correct solution to a particular problem 137 minutes into the contest after having twice submitted incorrect programs for that problem, 137 + 20 + 20 (i.e., 177) minutes are charged against the team. Note that no time is charged against a team for a problem that it never solved, regardless of how many submissions it may have made for that problem.

You are to develop a program that takes as input a log of all program submissions and that produces as output a "scoreboard" listing the teams in order of their ranks. Details follow.

Input:
The first line contains three nonnegative integers, k, m, and n, where k is the number of competing teams, m is the number of problems posed, and n is the number of submissions made during the contest. The teams are numbered from 1 to k and the problems are numbered from 1 to m. You may assume that $0 < k \leq 20$ and $0 < m \leq 6$. There is no *a priori* upper bound upon n, however.

Each of the next n lines of input represents a program submission (including the judge's verdict) and consists of four integers:

(1) a team number (a positive integer between 1 and k),

(2) a problem number (a positive integer between 1 and m),

(3) a time stamp (a nonnegative integer equal to the number of minutes that had elapsed between the time that the contest began and the time that the submission occurred), and

(4) a verdict (0 for incorrect, 1 for correct).

You may assume that the time stamps of the submissions are in nondecreasing order. That is, the time stamp in a submission can be no greater than the time stamp in any submission following it. You may also assume that, once a team has submitted a correct solution for a particular problem, it will never make another submission for that problem.

Output:
The first line should contain column headings for rank, team number, number of problems solved, and time charged. The second line should contain a horizontal line (comprised of dashes). The next k lines should include one line for each team, indicating its rank, team number, number of problems solved, and the time charged against it. The teams are to be listed in order according to their ranks. Two (or more) teams that are tied may be ranked in any order, relative to one another. (Notice teams 4 and 5 in the sample output below.)

Sample input:

```
5 6 26
2 4 34 0
3 2 39 0
3 1 51 0
3 1 60 1
2 4 65 0
2 3 87 0
2 1 93 0
2 3 102 0
2 3 139 0
2 4 143 0
3 2 150 0
1 6 150 0
3 2 153 1
3 3 160 0
2 6 167 0
2 4 169 1
3 3 175 1
2 3 180 0
2 3 211 1
2 1 230 0
5 2 245 0
3 6 268 1
1 6 270 1
1 4 281 0
1 2 285 0
1 4 287 1
```

Corresponding output:

Rank	Team	Problems Solved	Time Charged
1	3	4	736
2	2	2	520
3	1	2	597
4	5	0	0
5	4	0	0

University of Scranton
ACM Student Chapter / Computing Sciences Department
11th Annual High School Programming Contest (2000)

Problem 4: Digit Necklaces

Consider the sequence of digits that begins with two specified digits, d_0 and d_1, and continues as follows: for $i > 1$, let $d_i = (d_{i-2} + d_{i-1}) \bmod 10$. (That is, to obtain the i-th digit in the sequence, for $i > 1$, we take the ones digit of the sum of the $(i-1)$-st and $(i-2)$-nd.

Interestingly, no matter which digits we choose for d_0 and d_1, it will always happen that, for some $j > 0$, $d_0 = d_j$ and $d_1 = d_{j+1}$. That is, the sequence is guaranteed to cycle back around to its first two digits.

For example, if we choose $d_0 = 7$ and $d_1 = 1$, we get the sequence

$$\langle 7\ 1\ 8\ 9\ 7\ 6\ 3\ 9\ 2\ 1\ 3\ 4\ 7\ 1 \cdots \rangle$$

Because twelve distinct pairs of adjacent digits occur in the sequence before it cycles back to its first two digits, 7 and 1, we say that the pair of digits $(7, 1)$ has period 12. (Of course, every other pair of adjacent digits occurring in the above sequence has period 12, too.)

Develop a program that, given as input a pair of digits, reports its period.

Input:
The first line contains a positive integer n indicating the number of pairs of digits that are to be processed. Each of the next n lines contains a pair of digits, separated by a space.

Output:
For each of the n pairs of digits given as input, there should be a line of output indicating its period.

Sample input:	Corresponding output:
4	12
7 1	4
2 6	1
0 0	12
9 7	

University of Scranton
ACM Student Chapter / Computing Sciences Department
11th Annual High School Programming Contest (2000)

Problem 5: Text Formatting

In ancient times, before the widespread availability of application software for word processing on PC's (e.g., Microsoft Word, Corel WordPerfect), there existed less sophisticated software packages (available mostly for mainframe computers) known as text formatters. A text formatter takes as input a file containing text (i.e., a sequence of characters, typically representing a term paper, a memorandum, or some such document) and produces as output a file containing the same text, more or less, but formatted in a way specified by the user. Commonly, the text formatter expected to find embedded within the text itself the "instructions" describing how it was to be formatted. (In part, web browsers are text formatters that process text that has been "marked up" with HTML "tags".)

You are to develop a progam that acts as a very primitive text formatter. In order to specify the program's intended behavior, it will be useful to define the term *word*. For our purposes, a word is a sequence of characters that

(a) contains no spaces (i.e., blanks), and
(b) appears either at the beginning of a line or immediately after a space, and
(c) appears either at the end of a line or immediately before a space.

For example, in the line of text

```
was noted, the frog croaked.  But he
```

each of "`was`", "`noted,`", "`the`", "`frog`", "`croaked.`", "`But`", and "`he`" is a word, but neither "`the frog`" nor "`noted`" is a word, the former because it includes a space and the latter because it is followed in the text by a non-space (namely, a comma).

As input, the program will be given a number m representing the desired width of the text (measured in characters), followed by the text itself. The text produced as output is to satisfy the following conditions:

(1) Viewed as a sequence of words, the output must be the same as the input.
(2) The number of spaces appearing between adjacent words on a line must be exactly one, unless the first of the two words ends with one of the characters '.', '!', or '?', in which case the two words must be separated by exactly two spaces.
(3) A space may not appear as the first character on a line.
(4) A word may not be split across two lines (i.e., with some of it appearing at the end of one line and the rest appearing at the beginning of the next line).
(5) Any character beyond the m-th on a line must be a space. (Note: This is slightly less strict than to say "No line may be longer than m characters in length" because it allows spaces to occur at positions $m + 1$, $m + 2$, etc., which may make it slightly easier to develop a correct

program.)
(6) Every line (except the last one) must contain the maximum number of words that will fit on it, while at the same time not violating any of the conditions above. That is, it must not be possible to move the first word on a line to the end of the previous line without violating any of the conditions above.

Input
The first line contains a positive integer m indicating the desired width of the output text. On the lines that follow is the text itself. You may assume that, with the exception of the end-of-line and end-of-file "characters", all characters occurring in the text are "printable" symbols that appear on a typical computer keyboard. For example, no "control characters" will occur, nor will the tab character. You may also assume that m is no less than the length of the longest word in the text. (If that weren't true, it would be impossible to produce correct output!)

Output
The text produced as output is to satisfy conditions (1) through (6) listed above.

Sample Input:

30
My thesis will be to develop an optical character
recognition system (OCRS) for 2-D cadastral maps utilizing
the software engineering principles taught in the
University of Scranton's Software Engineering Masters
program. The system will take as input a digital image of a
map produced by a scanner, and after processing produce as
output an ASCII file of the text found in the map. Can
this be done in real time? The process needed will require
the system to locate the text within the map and also to
classify it so it can be converted into its correct ASCII
representation.

Corresponding Output:

My thesis will be to develop
an optical character
recognition system (OCRS) for
2-D cadastral maps utilizing
the software engineering
principles taught in the
University of Scranton's
Software Engineering Masters
program. The system will take
as input a digital image of a
map produced by a scanner, and
after processing produce as
output an ASCII file of the
text found in the map. Can
this be done in real time?
The process needed will
require the system to locate
the text within the map and
also to classify it so it can
be converted into its correct
ASCII representation.

University of Scranton
ACM Student Chapter / Computing Sciences Department
11th Annual High School Programming Contest (2000)

Problem 6: Let's Go Bowling

In the popular game of bowling, a player throws a heavy ball down a long, narrow lane at the end of which stand ten wooden pins. The player's goal is to knock down all ten pins. If she succeeds in a single attempt, she is said to have made a *strike*. If, after her first attempt, one or more pins are left standing, she throws the ball again in order to knock down the remaining pins. If she succeeds, she is said to have made a *spare*. In any case, after throwing two balls (or only one, if it resulted in a strike), the player is said to have completed a *frame*. If pins remain standing after the player has completed a frame, the frame is said to be *open*.

A game of bowling consists of ten frames. To calculate a player's score for a game, we award a certain number of points[1] for each frame, according to the following rules:

1. For an open frame, the number of points awarded is the number of pins knocked down in that frame.
2. For a spare, the number of points awarded is ten (10), plus the number of pins knocked down on the next ball thrown (i.e., the first ball in the next frame).
3. For a strike, the number of points awarded is ten (10), plus the number of pins knocked down on the next two balls thrown. (If a strike is not thrown in the next frame, these are the two balls thrown in the next frame; otherwise, these are the first balls thrown in each of the next two frames).

The score for a game is the sum of the points awarded in each of the ten frames.

In order to apply the above scoring method to the tenth (i.e., last) frame, sometimes the player must throw "extra" balls after completing that frame. Specifically, if she makes a spare in the tenth frame, ten pins are set up and she is given one more throw. If she makes a strike in the tenth frame, ten pins are set up and she is given two more throws. (If, on the first throw, she knocks down all ten pins, ten more are set up for the second throw.)

You are to develop a program that, given as input the number of pins knocked down by each throw in a game of bowling, calculates the score achieved by the player.

Input

The first line contains a positive integer n indicating how many games are to be scored. The following $2n$ lines describe the games, two lines per game. The first line describing a game contains a single positive integer m indicating the number of throws made during the game. The second line contains a sequence of m nonnegative integers indicating the number of pins

---

[1] In standard bowling terminology, we say *pins* rather than *points*, but here we will use the latter in order to prevent the former from taking on two different meanings.

knocked down on each of the throws in that game. There is nothing in the input to indicate where one frame ends and another begins.

You may assume that every game given as input is "valid". For example, every game described will include ten complete frames (plus any required extra throws after the tenth frame) and in no frame will more than ten pins be knocked down.

Output
For each game given as input, the output should show the corresponding final score, one per line.

```
Sample Input:                                  Corresponding Output:
-------------                                  ---------------------
3                                              300
12                                             179
10 10 10 10 10 10 10 10 10 10 10 10            211
17
9 1 10 6 2 10 10 8 2 9 1 10 9 0 9 1 8
17
10 9 1 8 0 7 3 10 10 10 9 1 8 2 10 10 8
```