University of Scranton
ACM Student Chapter / Computing Sciences Department
12th Annual High School Programming Contest (2002)

Problem 1: Fibonacci Sequences

A fibonacci sequence is an (infinite) sequence of numbers such that, beginning with the third element, each is the sum of the previous two. Hence, all the elements in a fibonacci sequence are determined by the first two. The fibonacci sequence most often cited is the one whose first two elements are both one. Its initial segment of length eleven (i.e., the sequence comprised of its first eleven elements) is as follows:

1 1 2 3 5 8 13 21 34 55 89

Develop a program that, given as input two integers i and j and a positive integer k, generates as output the initial segment of length k of the fibonacci sequence whose first two elements are i and j, respectively.

Input
The first line will contain a positive integer m indicating how many instances of the problem are to be solved. On each of the subsequent m lines, a single instance will be described by means of three numbers: i, j, and k as described above.

Output
For each instance of the problem given as input (as described by i, j, and k), print the first k elements of the fibonacci sequence whose first two elements are i and j, respectively.

Sample input

```
3
5 2 10
-3 4 14
0 -1 8
```

Corresponding output

```
5 2 7 9 16 25 41 66 107 173
-3 4 1 5 6 11 17 28 45 73 118 191 309 500
0 -1 -1 -2 -3 -5 -8 -13
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
12th Annual High School Programming Contest (2002)

Problem 2: Durations in Normal Form

An expression of the form

W Days, X Hours, Y Minutes, Z Seconds

(where W, X, Y, and Z are nonnegative integers) describes a duration. Such an expression is said to be in *normal form* if $W > 0$, $0 < X < 24$, $0 < Y < 60$, and $0 < Z < 60$. For example, each of the following is in normal form:

5 Days, 16 Hours, 12 Minutes, 53 Seconds
13 Days, 45 Minutes, 16 Seconds
21 Hours, 8 Minutes
36 Minutes

Notice how we omit mention of any unit of measure that is not needed (because the corresponding number of units is zero). In the third expression above, for instance, we mention neither Days nor Seconds.

In contrast to the above, none of the following expressions is in normal form, the first because the number of Hours exceeds 23, the second because the number of seconds exceeds 59, and the third because Hours are mentioned when it need not be.

5 Days, 49 Hours, 33 Minutes
2 Days, 7 Hours, 5 Minutes, 93 Seconds
1 Days, 0 Hours, 51 Minutes, 4 Seconds

You are to develop a program that, given as input a duration (in the form of a sequence of four nonnegative integers representing numbers of days, hours, minutes, and seconds, respectively), produces as output the same duration but expressed in normal form.

Input
The first line will contain a positive integer m indicating how many durations are to be processed. Each of the next m lines will contain a description of a duration in the form of four nonnegative integers (separated by one or more spaces) representing numbers of days, hours, minutes, and seconds, respectively.

Output
For each duration given as input, the program should produce three lines of output. The first

line echoes the duration given as input. The second line expresses the same duration, but in normal form. The third line is blank.

Sample input

```
4
5 49 33 0
2 7 5 3573
1 0 67 120
5 2 43 15
```

Corresponding output

```
Duration 1:  5 Days, 49 Hours, 33 Minutes, 0 Seconds
Normal Form:  7 Days, 1 Hours, 33 Minutes

Duration 2:  2 Days, 7 Hours, 5 Minutes, 3573 Seconds
Normal Form:  2 Days, 8 Hours, 4 Minutes, 33 Seconds

Duration 3:  1 Days, 0 Hours, 67 Minutes, 120 Seconds
Normal Form:  1 Days, 1 Hours, 9 Minutes

Duration 4:  5 Days, 2 Hours, 43 Minutes, 15 Seconds
Normal Form:  5 Days, 2 Hours, 43 Minutes, 15 Seconds
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
12th Annual High School Programming Contest (2002)

Problem 3: Smallest Two-Distinct-Digits Multiple

For every positive integer n, there exists a positive multiple of n whose decimal representation contains exactly two distinct digits. Develop a program that, given as input a positive integer n, outputs the smallest such multiple of n.

Input
The first line will contain a positive integer m indicating how many numbers are to be processed. Each of the next m lines will contain a positive integer.

Output
For each number given as input, three lines of output should be generated. The first echoes the number, the second indicates the smallest positive multiple of that number containing exactly two distinct digits, and the third is blank.

Sample input

```
4
125
22
755
103
```

Corresonding output

```
125
500

22
110

755
755

103
515
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
12th Annual High School Programming Contest (2002)

Problem 4: Occurrence-Counting Fixed Points

Let us define the *occurrence-counting operation* as follows: when applied to a sequence S of integers, this operation yields the sequence T obtained by replacing each value in S by the number of occurrences of that value in S.

For example, suppose we have the sequence

$$S_0 = \langle 2, 1, 2, -3, 5, 1, 5, 2, 9, 12 \rangle$$

Applying the occurrence-counting operation to it, we get the sequence

$$S_1 = \langle 3, 2, 3, 1, 2, 2, 2, 3, 1, 1 \rangle$$

Notice that wherever S_0 contains 2, S_1 contains 3, due to the fact that 2 occurs three times in S_0. Similarly, wherever S_0 contains 1, S_1 contains 2, because 1 occurs twice in S_0.

Applying the operation to S_1 yields

$$S_2 = \langle 3, 4, 3, 3, 4, 4, 4, 3, 3, 3 \rangle$$

Applying the operation to S_2 yields

$$S_3 = \langle 6, 4, 6, 6, 4, 4, 4, 6, 6, 6 \rangle$$

If we apply the operation to S_3, the resulting sequence is S_3 once again (because, in S_3, 6 occurs six times and 4 occurs four times!). A sequence such as this (i.e., one that yields itself when the occurrence-counting operation is applied to it) is said to be a fixed point (of that operation).

It is a fact that, if, as illustrated above, we repeatedly apply the occurrence-counting operation (starting with some sequence S), we will eventually obtain a sequence that is a fixed point.

Develop a program that takes as input a sequence S of integers and produces as output the fixed point obtained by repeatedly applying the occurrence-counting operation, starting with S. The output should also report the number of applications of the operation needed to arrive at that fixed point (for the first time).

Input
The first line will be a positive integer m indicating the number of instances of the problem that are to be solved. The following $2m$ lines contain the m instances, with each instance described on two lines. The first line of each instance gives the length n ($0 < n \leq 40$) of the sequence to be analyzed. The second line contains the sequence.

Output

For each instance, four lines of output should be generated. The first echoes the input sequence. The second gives the fixed point obtained from that sequence via repeated application of the occurrence-counting operation. The third reports how many applications of the operation were necessary in order to arrive at that fixed point. The fourth line is blank.

Sample Input

```
8
3 3 4 1 4 4 3 4
5
8 2 4 3 9
10
4 5 -1 -1 4 5 12 3 5 4
```

Corresponding output

```
Sequence 1:  3 3 4 1 4 4 3 4
Fixed Point:  3 3 4 1 4 4 3 4
Apps:  0

Sequence 2:  8 2 4 3 9
Fixed Point:  5 5 5 5 5
Apps:2

Sequence 3:  4 5 -1 -1 4 5 12 3 5 4
Fixed Point:  6 6 4 4 6 6 4 4 6 6
Apps:  3
```

University of Scranton
ACM Student Chapter / Computing Sciences Department
12th Annual High School Programming Contest (2002)

Problem 5: Area Covered by Overlapping Rectangles

Develop a program that, given as input a set of rectangles lying on the cartesian plane, reports the total area covered by those rectangles. (This is not as simple as finding the sum of the areas of the given rectangles, because the area of any section of the plane covered by two or more rectangles should be counted only once.)

In order to make the problem less difficult, you may make three assumptions about the rectangles given as input:

1. They all lie entirely on the finite sub-plane bounded by the lines $x = 0$, $x = 100$, $y = 0$, and $y = 100$.
2. All their sides are either parallel or perpendicular to the line $x = 0$.
3. All their corners correspond to points whose x- and y-coordinates are integers.

Input:
The first line will contain a positive integer m indicating how many instances of the problem are to be solved. Subsequent lines encode the instances. The first line of each instance contains a positive integer n indicating the number of rectangles to be processed. Each of the next n lines contains a description of one of those rectangles. Such a description contains four integers (all between 0 and 100, inclusive) x_1, y_1, x_2, and y_2, with $x_1 < x_2$ and $y_1 > y_2$. That is, (x_1, y_1) is the point corresponding to the *upper left* corner of the rectangle and (x_2, y_2) is the *lower right* corner.

Output:
For each instance, report the area covered.

Sample input

```
2
1
5 9 8 2
3
0 4 2 2
5 9 8 2
1 5 6 3
```

Corresponding output

```
Instance 1:  21 square units
Instance 2:  32 square units
```

Problem 6: Bridge Hand Evaluation

In a standard deck of playing cards, there are 52 cards, each one uniquely identified by its suit (either Spades, Diamonds, Hearts, or Clubs) and its rank (any of 2, 3, 4, ..., 10, Jack, Queen, King, or Ace).

In many card games, each player is dealt a hand (a collection of cards from the deck) and plays it via a sequence of tricks, each one either a winning or losing trick.

In the card game Bridge, each player evaluates her hand of thirteen cards prior to bidding against the other. The most popular method of hand evaluation is called the Milton-Work evaluation method. In this method, each Ace found in the player's hand is worth four points, each King is worth three points, each Queen is worth two points, and each Jack is worth one point.

Another method is known as the Losing Trick Count method, which works by estimating the number of losing tricks, or losers, that the player will suffer in playing her hand. For each of the four suits, we count the number of losers for that suit. If a hand contains no cards from a suit, count no losers for that suit. If a hand contains exactly one card from a suit, count one loser if it is not the Ace and zero losers otherwise. If a hand contains exactly two cards from a suit, the number of losers for that suit corresponds to how many of the two ranks Ace and King are not among the ranks of those two cards. If a hand contains three or more cards from a suit, the number of losers for that suit corresponds to how many of the three ranks Ace, King, and Queen are not among the ranks of those (three or more) cards.

Input:
The first line of input will be a positive integer m indicating the number of Bridge hands to evaluate. Each of the subsequent m lines will contain a description of a Bridge hand. In describing a Bridge hand, one character will be used for representing each card. Specifically, Ace is denoted by 'A', King by 'K', Queen by 'Q', Jack by 'J', and Ten by 'T'. Smaller ranks are denoted by the corresponding digits. That is, Nine is denoted by '9', Eight by '8', etc. Each hand will be sorted into four suits; within each suit, the cards will be sorted by rank from the Ace downward to the two. A space will separate the cards in one suit from those in the next. If a suit does not contain any cards, the suit will be represented by a dash (`-`).

Output:
The program will output the hand, followed by the Milton-Work point count and the Losing Trick Count losers.

Sample input

```
2
AK A54 T9432 KJ5
KQ5432 J76 - K854
```

Corresponding output

```
Hand 1:  AK A54 T9432 KJ5
Milton-Work Points:  15
Losing Tricks:  7

Hand 2:  KQ5432 J76 - K854
Milton-Work Points:  9
Losing Tricks:  6
```