University of Scranton
ACM Student Chapter / Computing Sciences Department
21st Annual High School Programming Contest (2011)

Problem 1: Arithmetic Progression Generation

An *arithmetic progression* is a sequence of numbers of the form

$$\langle m, m+k, m+2k, m+3k, \cdots, m+(r-1)k \rangle$$

Notice that every pair of adjacent elements differs by the same value (namely, k). We refer to m, k, and r, respectively, as the *initial element*, *common difference*, and *length* of the sequence.

Develop a program that, given as input integers m and k and a positive integer r, displays the arithmetic progression with initial element m, common difference k, and length r.

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 by three integers representing m, k, and r as described above, with r necessarily greater than zero.

Output: For each instance of the problem given as input, display on a single line the elements in the arithmetic progression that it describes.

```
Sample input:              Resultant output:
------------               ----------------
3                          0 1 2 3 4 5 6 7
0 1 8                      5 2 -1 -4 -7
5 -3 5                     7 7 7 7
7 0 4
```

Problem 2: Fractions in Simplest Form

A *fraction* is a ratio between two integers p and q ($q \neq 0$), typically written as p/q.

We say that the fraction p/q is in *simplest form* if and only if $q > 0$ and the greatest common divisor of p and q is 1. (*Note:* Every non-zero integer is a divisor of zero.)

Two fractions p/q and r/s are equal if and only if $ps = qr$. It turns out that for every fraction there is a unique fraction that is both equal to it and in simplest form.

Develop a program that, given two integers p and q, where $q \neq 0$, outputs the unique fraction that is equal to p/q and is in simplest form.

Input: The first line contains a positive integer n indicating how many fractions are to be put into simplest form. Each of the following n lines contains two integers p and q, with $q \neq 0$, and represents the fraction p/q.

Output: For each fraction provided as input, it and its simplest form are to be displayed, separated by two spaces and an equal sign. (See sample output below for proper formatting.)

```
Sample input:                   Resultant output:
-------------                   -----------------
5                               32/-6 = -16/3
32 -6                           0/-13 = 0/1
0 -13                           252/1560 = 21/130
252 1560                        -462/-910 = 33/65
-462 -910                       45/14 = 45/14
45 14
```

Problem 3: String Editor

A *string* is a sequence of characters, the positions of which are numbered from zero up to, but not including, the length of the string. For example, in the string `computer`, which has length eight, `c` occurs at position zero and `t` occurs at position five. Within a string are *substrings*, which we identify by length and starting position. For instance, within `computer` the substring of length four starting at position two is `mput`.

There are several operations by which a string S can be edited. For example, a particular substring within S can be either *removed*, *reversed*, or *replaced* by a specified string. Or we can *insert* a specified string at a specified position within S. Or we can *prepend* (attach at the front) or *append* (attach at the end) a specified string to S.

Develop a program that applies a succession of string-editing operations, which are provided as input, to a string S (which we view as being mutable). Below is described the syntax of each kind of operation, and the effect it should have upon the value of S. For convenience, we use K to refer to the length of S. Initially, S should be the empty string (i.e., the string having length zero).

In describing the syntax of an editing operation, we use *str* to refer to a string that acts as an operand of that operation, and we use m and n to refer to integer operands. In several operations, m and n are used for identifying a substring of S, with m being its starting position and n being its length. You may assume that m and n will be nonnegative. Also, for such operations, in the case that $m \geq K$ the operation is to have no effect. In the case that $m < K < m + n$, the operation should behave as though $m + n = K$. (Recall that K is the length of S.) If the described effects of an operation are not clear, the sample input/output (shown at the end) should clarify it.

New:
Syntax: N *str*
Effect: changes S to be the same as *str*.

Display: (version 1)
Syntax: D
Effect: displays S (in its entirety).

Display: (version 2)
Syntax: D m n
Effect: displays the substring of S be beginning at position m and having length n.

Append:
Syntax: A *str*
Effect: appends *str* to the end of S.

Prepend:
Syntax: P *str*
Effect: prepends *str* to the beginning of S.

Insert:
Syntax: I *m* *str*
Effect: inserts *str* within S at position m. (If $m > K$, treat it as though $m = K$.)

Remove:
Syntax: R *m* *n*
Effect: removes from S the substring of length n beginning at position m.

Replace:
Syntax: L *m* *n* *str*
Effect: replaces the substring of length n starting at position m by *str*.

Reverse:
Syntax: V *m* *n*
Effect: within S, reverses the substring of length n starting at position m.

Quit:
Syntax: Q
Effect: execution terminates.

That concludes the description of the various operations.

Input: The input is a sequence of string-editing operations, of the kinds described above, one per line.

Output: For each occurrence of a **Display** operation (of which there are two variations), the program should output, on a single line, the specified string (i.e., either S or some substring thereof).

```
Sample input:               Resultant output:
------------                ----------------
N abcdefg                   abcdefg
D                           cde
D 2 3                       aedcbfg
V 1 4                       aedcbfgxyz
D                           ABCaedcbfgxyz
A xyz                       ABCaedcb
D                           bcdeaCBA
P ABC                       bcdXYZuvwBA
D                           gorn
R 8 10
D
V 0 17
D
L 2 4 XYZuvw
D
N gorn
D
Q
```

Problem 4: Arithmetic Progressions

An *arithmetic progression* is a sequence of numbers of the form

$$\langle m, m+k, m+2k, m+3k, \cdots, m+(r-1)k \rangle$$

Notice that every pair of adjacent elements differs by the same value (namely, k). We refer to m, k, and r, respectively, as the *initial element*, *common difference*, and *length* of the sequence.

Develop a program that, given as input an integer s and positive integers k and r, identifies an arithmetic progression having an initial element that is an integer, having common difference k, having length r, and whose elements sum to s, if such a sequence exists. If no such sequence exists, the program should report that fact.

Hint: Where p is a positive integer,

$$1 + 2 + 3 + \cdots + (p-2) + (p-1) = \frac{p^2 - p}{2}$$

Input: The first line contains a positive integer n indicating how many instances are subsequently described. Each instance is described on a single line by three positive integers s, k, and r.

Output: For each instance given as input, as described by three integers s, k, and r, the program should output either the initial (integer-valued) element of an arithmetic progression whose elements sum to s, whose common difference is k, and whose length is r, or else it should report that no such arithmetic progression exists. See the sample output for the precise format.

```
Sample input:          Resultant output:
-------------          -----------------
4                      18
100 1 5                none
100 1 4                -36
-63 9 7                7
-2 -5 4
```

Problem 5: Transitivity Detection

In mathematics, a *binary relation over* (a set) S is a set of ordered pairs each component of which is a member of S. For example, $R_1 = \{(a,a),(a,b),(a,d),(b,c),(c,b),(c,c),(d,a)\}$ is a binary relation over the set $\{a,b,c,d\}$.

Although you may never have thought about it in this way, $<_{\mathcal{N}}$ (less than over the natural numbers) is a binary relation, as

$$<_{\mathcal{N}} = \{(0,1),(1,2),(2,3),\ldots,(0,2),(1,3),(2,4),\ldots,(0,3),(1,4),(2,5),\ldots\}$$

That is, for natural numbers j and k, (j,k) is a member of $<_{\mathcal{N}}$ if and only if j is less than k. In other words, the expression $j < k$ is just an abbreviation for $(j,k) \in <_{\mathcal{N}}$!

Let R be a binary relation over S. We say that R is *transitive* if, for every $x,y,z \in S$, $(x,y) \in R$ and $(y,z) \in R$ implies $(x,z) \in R$.

As you should know, $<_{\mathcal{N}}$ is transitive, because (for any natural numbers j, k, and m) $j < k$ and $k < m$ implies $j < m$.

To demonstrate that a particular binary relation is *not* transitive, it suffices to identify a single counterexample to the definition of transitivity. For example, to show that R_1 (from above) is not transitive, it suffices to point out that even though (a,b) and (b,c) are members of R_1, (a,c) is not. (Alternatively, we could have pointed out that (b,c) and (c,b) are members of R_1, but (b,b) is not. And these are not the only two counterexamples.)

Develop a program that, given (a tabular representation of) a binary relation over the set $\{0,1,\ldots,m-1\}$, for some $m > 0$, reports whether or not it is transitive, and, in the case that it is not, identifies one counterexample.

Input: The first line contains a positive integer n indicating how many binary relations are to be analyzed. Each relation is given by a positive integer m appearing by itself on a line, followed on the next m lines by an $m \times m$ table describing the relation. Each entry in the table is either 1 or 0, and adjacent entries in a row are separated by a space. A one in the i-th row and j-th column means that (i,j) *is* a member of the described relation. A zero means that it *is not*. Rows and columns are assumed to be numbered beginning at zero going from top to bottom and from left to right, respectively; hence the described relation is over the set $\{0,1,\ldots,m-1\}$.

Output: For each binary relation provided as input, report whether or not it is transitive. In the case that it is not, identify three values x, y, and z such that (x,y) and (y,z) are members of the relation, but (x,z) is not.

```
Sample input:             Explanation:
------------              ------------
4                         four binary relations are to be analyzed
4                         the first relation is over the set {0,1,2,3}
0 1 1 1                   (0,1), (0,2), and (0,3) are members
0 0 1 1                   (1,2) and (1,3) are members
0 0 0 1                   (2,3) is a member
0 0 0 0                   there are no members of the form (3,y)
2                         the second relation is over the set {0,1}
0 1                       (0,1) is a member
1 1                       (1,0) and (1,1) are members
3                         the third relation is over the set {0,1,2}
0 0 0                     there are no members of the form (0,y)
0 0 0                     there are no members of the form (1,y)
0 0 0                     there are no members of the form (2,y)
5                         the fourth relation is over the set {0,1,2,3,4}
1 0 1 1 0                 (0,0), (0,2), and (0,3) are members
0 0 0 0 0                 there are no members of the form (1,y)
0 1 1 1 0                 (2,1), (2,2), and (2,3) are members
1 0 0 0 1                 (3,0) and (3,4) are members
0 1 0 0 0                 (4,1) is a member


Resultant output:
-----------------
transitive
non-transitive: 0 1 0
transitive
non-transtive: 3 4 1
```

Problem 6: Logical Consistency of Liar, Liar Input Data

A recent programming puzzle on Facebook, "Liar, Liar", describes a scenario in which there is a group of people, each of whom is either "unfailingly honest" (i.e., a non-liar) or "compulsively deceptive" (i.e., a liar). Furthermore, each member of the group knows into which of these two categories each of the others belongs. Each member of the group is asked to provide a list of persons who are liars. (Of course, a liar, being such, will place only non-liars on his list! A non-liar's list, on the other hand, will be accurate.)

Facebook invites you to submit a program that, using such lists as input data, reports the sizes of (i.e., number of people in) each of the two categories, but without indicating which size matches which category. (It turns out that, in some cases, there is no way to determine these sizes. However, that is of no concern to us here.)

As an example, suppose that Ann identifies Bill and Carol as liars. If Ann is a non-liar, then, indeed, Bill and Carol must be liars. On the other hand, if Ann is a liar, then neither Bill nor Carol is one! At this point, all we can conclude is that, among the three persons mentioned, one of them (Ann) is in one category and two of them (Bill and Carol) are in the other.

If, in addition to Ann claiming that Bill and Carol are liars, Bill (and/or Carol) identifies Ann as a liar, that doesn't really give us any new information, as it is simply consistent with Ann being in one category and both Bill and Carol being in the other.

Suppose, however, that (in addition to Ann's claims) Bill identifies Carol as a liar (or vice versa). Then we have a logical inconsistency, because Ann's claims imply that Bill and Carol are in the same category whereas Bill's claim implies that Bill and Carol are in different categories.

To analyze it in a slightly different way, suppose that Ann is a non-liar. She names Bill as a liar, so he must be one. Therefore, Bill's claim that Carol is a liar is false, making her a non-liar. But Ann mis-identifies Carol as a liar, making Ann a liar, which contradicts our assumption! Had we instead assumed Ann to be a liar, a similar analysis would have led us to the contradictory conclusion that she is a non-liar.

In our example above, an inconsistency arises as a result of the interaction of three people's claims. In a more complicated case, it may be necessary to take into account the claims of five, or seven, or more people in order to ascertain that an inconsistency is present.

Develop a program that, given as input the "liar list" of each member of a group, reports whether the given data are logically consistent.

Input: The first line contains a positive integer m indicating how many instances of the problem are described thereafter. Each instance of the problem is described using $n + 1$ lines, where n, which indicates the number of people in the group, is the positive integer appearing on the first such line. (For the programmer's convenience, the people are identified not by names but rather by the nonnegative integers 0 through $n - 1$.) On the n lines that follow are the "liar lists" of the group members, one list per line, with the first list being that of person 0, the next being that of person 1, etc., etc. Each list begins with a nonnegative integer k indicating the length of the list followed by k nonnegative integers in increasing order, each in the range $0..n-1$. Of course, each value in a list identifies a person who is claimed (by the "author" of the list) to be a liar.

Output: For each instance of the problem, the output generated should be a one-line message indicating whether or not the input data is logically consistent. See sample output below for the exact forms of the two possible messages.

```
Sample Input:       Explanation:
-------------       ------------
3                   three instances of the problem are described
4                   first instance has a group of four persons
2 1 2               person 0 claims that persons 1 and 2 are liars
0                   person 1 makes no claims about liars
1 0                 person 2 claims that person 0 is a liar
0                   person 3 makes no claims about liars
5                   second instance has a group of five persons
1 1                 person 0 claims that person 1 is a liar
2 0 1               person 1 claims that persons 0 and 1 are liars
3 1 3 4             person 2 claims that persons 1, 3, and 4 are liars
0                   person 3 makes no claims about liars
1 2                 person 4 claims that person 2 is a liar
7                   third instance has a group of seven persons
1 5                 person 0 claims that person 5 is a liar
1 6                 person 1 claims that person 6 is a liar
2 1 4               person 2 claims that persons 1 and 4 are liars
2 0 6               person 3 claims that persons 0 and 6 are liars
2 2 5               person 4 claims that persons 2 and 5 are liars
0                   person 5 makes no claims about liars
2 1 5               person 6 claims that persons 1 and 5 are liars
```

```
Resultant output:
-----------------
Consistent
Not consistent
Not consistent
```