University of Scranton
ACM Student Chapter / Computing Sciences Department
14th Annual High School Programming Contest (2004)

Problem 1: Two Smallest and Two Largest

Develop a program that, given as input a sequence of integers, identifies the two smallest and two largest (distinct) values appearing in the sequence. You may assume that the sequence includes at least two distinct values.

Input: The first line contains a positive integer n indicating how many sequences are to be processed. Each sequence is described by a positive integer m indicating its length, followed by its m elements, one per line. You may assume that, among the elements in a sequence, there are at least two different values.

Output: For each sequence given as input, three lines of output are to be generated. The first identifies the two smallest (distinct) values occurring in the sequence, in ascending order; the second line identifies the two largest (distinct) values occurring in the sequence, in descending order. See sample output for correct format.

```
Sample Input:                Corresponding Output:
-------------                ---------------------
2                            Two Smallest: -4 1
5                            Two Largest: 1 -4
-4
-4                           Two Smallest: -6 0
-4                           Two Largest: 33 8
1
-4
10
5
8
2
-6
0
4
2
0
33
4
```

Problem 2: Grade Point Averages

Develop a program that, given as input the grades that a student earned in courses taken over the span of (possibly) several semesters, outputs relevant statistics (e.g., grade point average (GPA)) about her performance in each semester and in the aggregate.

The grades that can be earned in a course, together with their respective *point values* are

```
            B+: 3.3     C+: 2.3     D+: 1.3
A : 4.0     B : 3.0     C : 2.0     D : 1.0     F : 0.0
A-: 3.7     B-: 2.7     C-: 1.7
```

Depending upon how many hours per week a course meets, it is worth either 1, 2, 3, or 4 credits. If a student earns a grade of D or better (i.e., anything but F) in a course, she earns the corresponding number of credits. Otherwise, she fails to earn any credits.

The number of *quality points* that a student earns from taking a course is the product of its number of credits with the point value of the student's grade in the course. For example, if a student earns B- in a 3-credit course, she earns $3 \cdot 2.7$, or 8.1, quality points.

A student's semester GPA is found by dividing the sum of the credits of the courses taken by the student during that semester into the sum of the quality points she earned in those courses.

A student's aggregate GPA is found by dividing the sum of the credits of all the courses taken by the student into the quality points that she earned in all those courses.

Input: The first line contains a positive integer n indicating the number of semesters for which we have data. The data for each semester begins with a line containing a positive integer m indicating in how many courses the student was enrolled during that semester. Each of the following m lines contains an integer between 1 and 4, inclusive, indicating the number of credits of one of those courses, followed (on the same line) by the grade that the student earned in that course.

Output: For each semester's worth of data, six lines of output are to be generated. The first one identifies the semester by number (beginning with 1). The second line reports how many credits the student attempted to earn that semester. The third line reports how many credits the student succeeded in earning that semester. The fourth line reports how many quality points the student earned that semester. The fifth line reports the student's GPA for that semester. The sixth line is blank.

Following the output for the last semester, the corresponding aggregate statistics are reported, in a similar format. (See the sample output below.)

Sample Input:

```
2
5
3 B
3 C+
4 B+
3 A
3 B
4
3 F
3 A
3 B
4 B-
```

Corresponding Output:

```
Semester 1:
  Credits Attempted: 16
  Credits Earned: 16
  Quality Points: 50.1
  GPA: 3.13

Semester 2:
  Credits Attempted: 13
  Credits Earned: 10
  Quality Points: 31.8
  GPA: 2.45

Aggregate:
  Credits Attempted: 29
  Credits Earned: 26
  Quality Points: 81.9
  GPA: 2.82
```

Problem 3: Point on a Line Segment?

Develop a program that, given as input a point P and the endpoints P_1 and P_2 of a line segment, determines whether the former lies on the latter. You may assume that no two among P, P_1, and P_2 are the same point (i.e., they are three distinct points). All of this is in the context of the cartesian plane.

Input: The first line contains a positive integer n indicating how many instances of the problem are to follow. Each instance is described on one line containing six real numbers: the first two identify P, the next two identify P_1, and last two identify P_2. Each point is identified by its x and y coordinates, respectively.

Output: For each given instance of the problem, one line of output is to be generated. That line reports whether or not P lies on the line segment (P_1, P_2) (i.e., the line segment with endpoints P_1 and P_2). See the examples below for the exact format.

Warning: The results of computer calculations on real numbers are only approximations. Hence, it is rarely a good idea to compare two real numbers a and b for equality. Rather, one should compare them to determine whether they are "almost equal" (see details below). If they are almost equal, we attribute any difference between them to errors introduced during calculation and we continue as though we had found them to be equal.

One standard approach is to deem a and b to be "almost equal" if the *relative error* between them is no greater than some pre-chosen, small value (usually called by the Greek letter ϵ). We define the relative error between a and b to be $\frac{|a-b|}{c}$, where c is the maximum of $|a|$ and $|b|$. (In the exceptional case in which $a = 0 = b$, the relative error between them is defined to be zero.) Use $\epsilon = 0.001$.

Hint: A necessary condition for P to lie on (P_1, P_2) is that P lies on the line passing through P_1 and P_2. But this is not a sufficient condition, as is evidenced by the first example below.

```
Sample Input:
-------------
3
0.0 6.0 2.0 4.0 5.0 1.0
0.0 2.5 0.0 -5.7 0.0 7.2
13.1 3.5 -0.5 0.5 3.0 2.5

Corresponding Output:
---------------------
(0.0,6.0) does not lie on line segment ((2.0,4.0), (5.0,1.0))
(0.0,2.5) lies on line segment ((0.0,-5.7), (0.0,7.2))
(13.1,3.5) does not lie on line segment ((-0.5,0.5), (3.0,2.5))
```

Problem 4: Hills and Plateaus

An *ascent* is a sequence of length two or more in which each element, except for the first, is larger than the one preceding it. An example of an ascent of length four is $\langle -3\ 2\ 4\ 5 \rangle$.

A *descent* is defined similarly, except that each element, except for the first, is smaller than the one preceding it. An example of a descent of length three is $\langle 5\ 2\ 0 \rangle$.

A *hill* is any sequence that is either an ascent or a descent.

A *plateau* is a sequence of length two or more in which each element, except for the first, is equal to the one preceding it. An example of a plateau of length three is $\langle 8\ 8\ 8 \rangle$.

A sequence can be broken up, in a unique way, into sub-sequences that are (unextendable) hills and plateaus. For example, consider the sequence

$$\langle -4\ 2\ 3\ 3\ 3\ 6\ 5\ 2\ 0\ 0\ 1\ 2\ 5\ 5 \rangle$$

If we enclose each ascent within parentheses (i.e., '(' and ')'), each descent within curly braces (i.e., '{' and '}'), and each plateau within square brackets (i.e., '[' and ']'), we get

$$\langle\ (-4\ 2\ [3)\ 3\ (3]\ \{6)\ 5\ 2\ [0\}(0]\ 1\ 2\ [5)\ 5]\ \rangle$$

Notice that, where a hill meets a plateau (or two hills meet), the last element of one is the first element of the other.

Develop a program that, given as input a sequence of integers, reports the number of ascents, descents, and plateaus within that sequence, as well as the lengths of the longest ascent, longest descent, and longest plateau.

Input: The first line contains a positive integer n indicating how many sequences are to be processed. Each sequence is described by a positive integer m on one line, followed by m lines, each containing one element of the sequence.

Output: For each sequence given as input, four lines of output are to be produced. The first line reports the number of ascents within the sequence and the length of the longest one(s). (See the sample output below for the required format.) Similarly, on the second and third lines, respectively, are reported the number and maximum lengths of the descents and plateaus in the sequence. The fourth line is to be blank.

```
Sample Input:              Corresponding Output:
------------               ---------------------
2                          # ascents: 3   max length: 4
14                         # descents: 1  max length: 4
-4                         # plateaus: 3  max length: 3
2
3                          # ascents: 0   max length: 0
3                          # descents: 1  max length: 2
3                          # plateaus: 0  max length: 0
6
5
2
0
0
1
2
3
3
2
13
10
```

Problem 5: Distance to a Cycle

Examine the figure below, which depicts a *directed graph* (or, more concisely, *digraph*) that we will refer to as G. A digraph is composed of *vertices* (the singular of which is "vertex") and *edges*. We use circles to depict vertices and arrows to depict edges. An edge goes from one vertex to another (possibly itself). It is customary to number the vertices (as in the figure) so as to identify them.

The *outdegree* of a vertex is the number of edges "leaving" it. In this problem, we will be concerned only with digraphs in which all vertices have outdegree 1. Notice that G has this property.

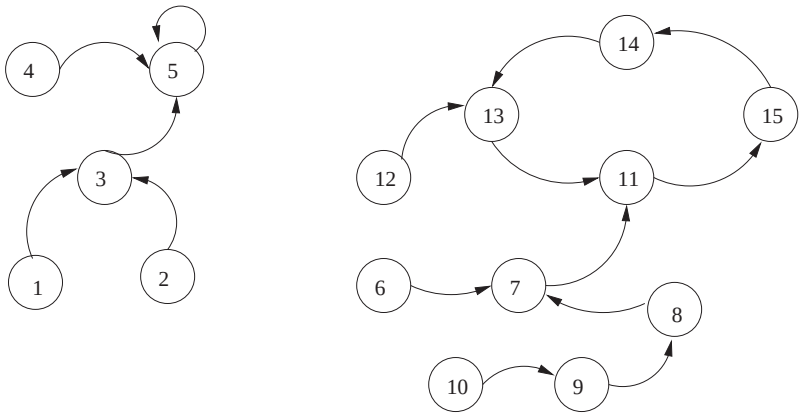

Figure 1: Directed Graph G

In a digraph, a *path* is a sequence of vertices such that there is an edge from each vertex to the next. The *length* of a path is the number of edges that are crossed in "walking" it. In G, for example, $\langle 8, 7, 11, 15\rangle$ is a path of length 3. (The length of a path is always one less than the length of the sequence of vertices describing it.)

A *cycle* is a path of length greater than zero that begins and ends with the same vertex. In G, for example, $\langle 14, 13, 11, 15, 14\rangle$ is a cycle, as is $\langle 5, 5\rangle$.

Observe that, in G, some vertices lie on cycles (e.g., 5, 14, 15) whereas other ones do not (e.g., 1, 3, 8, 10). However, for every vertex, there is a path beginning there that leads to a vertex that lies on a cycle. This is true not only of G, but of every digraph in which all vertices have outdegree greater than 0.

Develop a program that, given as input a digraph in which every vertex has outdegree 1, outputs, for each vertex, the shortest path beginning at that vertex and ending in a vertex that lies on a cycle. (For a vertex that lies on a cycle, the shortest such path has length zero!)

Note: A much more concise (but possibly harder to understand) description of the problem is as follows:

Develop a program that, given an integer $n > 0$ and a function $f : \{1, 2, \ldots, n\} \longrightarrow \{1, 2, \ldots, n\}$ determines, for each i in $\{1, 2, \ldots, n\}$, the minimum k for which, for some $j > 0$, $f^k(i) = f^{k+j}(i)$. (By $f^m(i)$ we mean $f(f(\cdots(f(i))\cdots))$, where there are m applications of f. More precisely, we define recursively $f^0(i) = i$ and $f^{m+1}(i) = f(f^m(i))$ for $m \geq 0$.)

Notice that there is an obvious interpretation of f as a digraph with vertices numbered 1 through n in which, for each i, the lone edge leaving vertex i goes to vertex $f(i)$. Under this interpretation, $f^m(i)$ is the vertex at the end of the path of length m beginning at vertex i.
End of note.

Input: The first line contains an integer n, $0 < n \leq 50$, indicating the number of vertices in the digraph. Following that are n lines, each containing an integer between 1 and n. The k-th such line identifies the vertex at the end of the outgoing edge from vertex k. (For example, if the 5-th line contains 13, that means that the edge leaving vertex 5 goes to vertex 13.)

Output: For each vertex, there is to be one line of output, which is to contain the sequence of vertices describing the shortest path from that vertex to one that lies on a cycle. The output for vertex 1 should precede that for vertex 2, which should precede that for vertex 3, etc.

Sample Input:	Corresponding Output:
15	1 3 5
3	2 3 5
3	3 5
5	4 5
5	5
5	6 7 11
7	7 11
11	8 7 11
7	9 8 7 11
8	10 9 8 7 11
9	11
15	12 13
13	13
11	14
13	15
14	