University of Scranton
ACM Student Chapter / Computing Sciences Department
18th Annual High School Programming Contest (2008)

Problem 1: Look-and-Say Sequences

Consider the sequence of integers

$$1,\ 11,\ 21,\ 1211,\ 111221,\ 312211,\ 13112221,\ 1113213211,\ \ldots$$

It is referred to as the **look-and-say sequence** with **seed** 1. The seed refers to the first element of the sequence. Each subsequent element in the sequence is generated from its predecessor as follows: group adjacent occurrences of the same digit and then use two digits to indicate, for each group, how many are in it and which digit it contains, respectively.

For example, the seed, 1, is read as "one 1", which translates to the second element, 11. We read that as "two 1's", which translates to the third element, 21. Moving forward to the fourth element, 1211, we read it as "one 1, then one 2, then two 1's", which translates to the fifth element, 111221.

Of course, each different choice of a seed results in a different sequence.

Develop a program that, given as input a nonnegative integer j and a positive integer k, produces as output the first k elements of the Look-and-Say sequence with seed j.

Input The first line contains a positive integer n indicating how many sequences are to be generated. Each of the next n lines contains two positive integers j and k that describe a sequence, as explained in the preceding paragraph.

Output For each Look-and-Say sequence described in the input, the program should generate as output a single line containing the elements of that sequence.

```
Sample input:            Resultant output:
-------------            -----------------
5                        1 11 21 1211 111221 312211 13112221 1113213211
1 8                      22 22 22 22 22 22
22 6                     55555333337 555317 35131117 131511133117
55555333337 4            0 10 1110 3110
0 4                      34
34 1
```

Problem 2: Histogram Generation

Develop a program that, given a collection of quiz scores, generates a histogram showing the collection's frequency distribution. Quiz scores will be in the range zero to twenty, inclusive.

Input: The first line contains a positive integer n indicating how many collections are to be processed. Each collection is described on two or more lines, the first of which contains the number of quiz scores m ($m > 0$) in that collection. The remaining lines contain the quiz scores themselves, twenty scores per line up until the last one, which will have a number of scores equal to the remainder obtained when m is divided by 20.

Output: For each collection of quiz scores, the program should generate a histogram indicating the frequency with which each of the values zero through twenty occurs in that collection.

The histogram's form should be as in the sample output below. Specifically, the only quiz scores that should be mentioned explicitly in the histogram are those between the minimum and maximum scores recorded, inclusive. (For the first example below, this range is ten through nineteen.)

The number of lines generated for a histogram should be exactly $k+3$, where k is the maximum number of occurrences of any of the quiz scores. This accounts for there being k asterisks in the tallest column, one line for the scale (containing dashes and plus signs), one line containing the quiz scores, and a blank line to separate one histogram from the next.

```
Sample input:
-------------
2
22
10 12 16 12 10 19 19 19 12 14 13 14 14 16 17 14 15 13 18 15
15 17
5
20 4 12 12 17

Resultant output:
-----------------
             *
       *     *  *           *
 *     *  *  *  *  *  *     *
 *     *  *  *  *  *  *  *  *
-+--+--+--+--+--+--+--+--+--+
10 11 12 13 14 15 16 17 18 19

                         *
 *                       *              *        *
-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
 4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Problem 3: Riemann Sums

In *Integral Calculus*, we study problems that can be solved by "finding the area under a curve". That is, we have a function f mapping real numbers to real numbers, a lower bound a, and an upper bound b, and we wish to determine the area of the region bounded on the left by the line described by the equation $x = a$, on the right by the line described by the equation $x = b$, below by the x-axis (i.e., the line described by the equation $y = 0$), and above by the curve corresponding to the function f.

Using techniques from Calculus, we can often find this area exactly. In those cases in which we cannot, we resort to computing an approximation to the area using what is called a Riemann sum.

We proceed by choosing a positive integer m and by placing m rectangles side by side, each of width $\Delta x = (b - a)/m$, between the lines $x = a$ and $x = b$. (Hence, the left edges of the rectangles lie on the lines $x = a$, $x = a + \Delta x$, $x = a + 2\Delta x$, $x = a + 3\Delta x$, etc.) The height of each rectangle is chosen so that its upper left corner is a point on the curve. We then calculate the sum of the areas of the rectangles, which provides us with an estimate of the area of the region of interest. (Generally speaking, the larger the value of m that we choose, the better we expect the estimate to be.)

As an example, see the figure below, in which $a = 0$, $b = 16$, $m = 8$, and f is given by the graph.

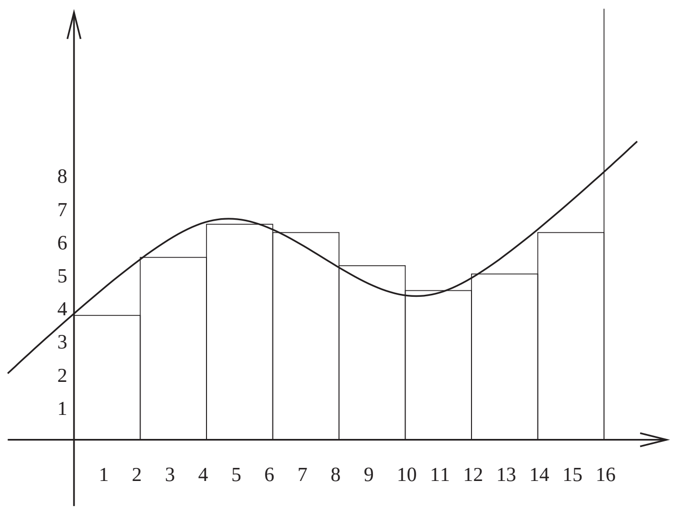

Figure 1: Estimating the Area Under a Curve by a Riemann Sum

Develop a program that, given real numbers a and b (with $a < b$), a positive integer m, and a description of a polynomial p (satisfying $p(x) \geq 0$ for all $x \in [a, b]$) estimates the area of the region bounded by $x = a$ on the left, $x = b$ on the right, $y = 0$ below, and p above, using the method described above.

Note that a polynomial $p(x)$ is simply a function that can be expressed by

$$p(x) \ = \ c_0 + c_1x + c_2x^2 + c_3x^3 + \ \cdots \ + c_kx^k$$

for some $k \geq 0$ and real numbers (called coefficients) c_0, c_1, ..., c_k. We say that k is the *degree* of p.

Input: The first line contains a positive integer n indicating the number of regions whose areas are to be estimated. On the following n lines are described the regions, one per line. Each one is described by real numbers a and b (the lower and upper bounds, respectively), a positive integer m (the number of rectangles to use in calculating the estimate of area), a nonnegative integer k (the degree of the polynomial), and the $k+1$ coefficients of the polynomial, beginning with c_0 and ending with c_k.

Output: For each region described, produce as output a two-line message identifying the polynomial, the bounds, the number of rectangles used in calculating the estimate of area, and the estimate itself. See the sample output below for the expected format. Put a blank line after each message. Note that computer arithmetic on real numbers is not exact, so don't be alarmed if your results to not exactly match those given in the sample output below.

```
Sample input:
-------------
3
0.0 4.0 4 1 0.0 2.0
0.0 4.0 8 1 0.0 2.0
-3.0 1.0 4 2 2.0 0.0 1.0


Resultant Output:
-----------------
Estimated area under y = 0.0x^0 + 2.0x^1
   between x = 0.0 and x = 4.0 using 4 rectangles is 12.0

Estimated area under y = 0.0x^0 + 2.0x^1
   between x = 0.0 and x = 4.0 using 8 rectangles is 14.0

Estimated area under y = 2.0x^0 + 0.0x^1 + 1.0x^2
   between x = -3.0 and x = 1.0 using 4 rectangles is 22.0
```

Problem 4: Next Permutation

A *permutation* of a set is simply a sequence that contains each member of that set exactly once. For example, $\langle 4,2,5,1,3\rangle$ is a permutation of the set $\{1,2,3,4,5\}$, as is $\langle 4,2,1,3,5\rangle$.

Let n be a positive integer, and let $S = \langle x_1, x_2, \ldots, x_n\rangle$ and $T = \langle y_1, y_2, \ldots, y_n\rangle$ be permutations of the set $\{1,2,\ldots,n\}$. Then we define $S < T$ (S is "less than" T) to mean that there exists k, $1 \leq k \leq n$, such that $x_k < y_k$ and $x_i = y_i$ for all i satisfying $1 \leq i < k$. In other words, $S < T$ means that, in the first position at which S and T disagree, S has the element of lesser value.

For example,

$$\langle 4,2,1,3,5\rangle < \langle 4,2,5,1,3\rangle$$

because in the first position where the two sequences disagree (namely, the third position), the permutation on the left has a smaller value (1) than does the permutation on the right (5).

It should be clear that, by this definition, we can list all the permutations of $\{1,2,\ldots,n\}$ in order, from least ($\langle 1,2,\ldots,n\rangle$) to greatest ($\langle n, n-1, n-2, \ldots, 1\rangle$).

As an example, here are the permutations of $\{1,2,3,4\}$ in order from least to greatest:

$\langle 1,2,3,4\rangle$, $\langle 1,2,4,3\rangle$, $\langle 1,3,2,4\rangle$, $\langle 1,3,4,2\rangle$, $\langle 1,4,2,3\rangle$, $\langle 1,4,3,2\rangle$, $\langle 2,1,3,4\rangle$, $\langle 2,1,4,3\rangle$, $\langle 2,3,1,4\rangle$, $\langle 2,3,4,1\rangle$, $\langle 2,4,1,3\rangle$, $\langle 2,4,3,1\rangle$, $\langle 3,1,2,4\rangle$, $\langle 3,1,4,2\rangle$, $\langle 3,2,1,4\rangle$, $\langle 3,2,4,1\rangle$, $\langle 3,4,1,2\rangle$, $\langle 3,4,2,1\rangle$, $\langle 4,1,2,3\rangle$, $\langle 4,1,3,2\rangle$, $\langle 4,2,1,3\rangle$, $\langle 4,2,3,1\rangle$, $\langle 4,3,1,2\rangle$, $\langle 4,3,2,1\rangle$

Develop a program that, given a positive integer n and a permutation S of the set $\{1,2,\ldots,n\}$, computes the smallest permutation T such that $S < T$.

Hint: Let $S = \langle x_1, x_2, \ldots, x_n\rangle$ and suppose that $x_j < x_{j+1} > x_{j+2} > x_{j+3} > \cdots > x_n$. (In other words, suppose that j is the last position in S at which an element is smaller than the next element.) Then the smallest permutation T greater than S is of the form $\langle x_1, x_2, \ldots, x_{j-1}, x_k, \ldots\rangle$, where x_k is the smallest element among $\{x_{j+1}, x_{j+2}, \ldots, x_n\}$ that is larger than x_j. *End of hint.*

Input: The first line contains the number $m > 0$ of permutations that will be provided in the remaining input. The next $2m$ lines contain those permutations, each one described on two lines, the first of which contains a positive integer n and the second of which contains the permutation itself (i.e., a sequence containing each member of $\{1,2,\ldots,n\}$ exactly once), with spaces separating its elements. You may assume that none of the permutations given is the largest one possible. (That is, none of them will be of the form $\langle n, n-1, \ldots, 1\rangle$.) You may also assume that $n \leq 30$.

Output: For each permutation S given as input, generate three lines of output: the first should display S, the second should display the smallest permutation that is larger than S, and the third should be blank.

```
Sample Input:
-------------
3
4
2 4 1 3
6
5 2 6 4 3 1
14
4 14 12 10 3 13 6 7 11 9 8 5 2 1


Resultant Output:
-----------------
2 4 1 3
2 4 3 1

5 2 6 4 3 1
5 3 1 2 4 6

4 14 12 10 3 13 6 7 11 9 8 5 2 1
4 14 12 10 3 13 6 8 1 2 5 7 9 11
```

Problem 5: Line Segment Rotation

Develop a program that, given as input the two endpoints P and Q of line segment PQ, outputs the endpoints of the line segment obtained from PQ by rotating it 90 degrees about its midpoint. In other words, the line segment obtained from PQ has the same length as PQ, is perpendicular to PQ, and has the same midpoint as PQ (at which they intersect).

Input: The first line contains a positive integer n indicating how many line segments are subsequently given. Each line segment is described on a single input line by its endpoints, each of which is identified by its x and y coordinates, respectively, which are real numbers.

Output: For each line segment given as input, the program generates a single line of output that identifies the endpoints of the given line segment as well as the line segment obtained by rotating the given line segment 90 degrees about its midpoint. See the examples below for the proper output format.

Numbers should be accurate to at least three digits, but note that computer arithmetic on real numbers is not exact, so don't be alarmed if your results do not exactly match the sample output below. Also note that the order in which the two endpoints appear in the result does not matter.

```
Sample input:
------------
5
-4.0 0.0 2.0 3.0
-3.7 -2.4 -3.7 6.2
4.5 5.0 0.5 -0.5
5.0 -12.0 -3.0 8.0
2.0 6.0 5.0 6.0


Resultant output:
-----------------
Rotating (-4.0,0.0)(2.0,3.0) yields (-2.5,4.5)(0.5,-1.5)
Rotating (-3.7,-2.4)(-3.7,6.2) yields (-8.0,1.9)(0.6,1.9)
Rotating (4.5,5.0)(0.5,-0.5) yields (-0.25,4.25)(5.25,0.25)
Rotating (5.0,-12.0)(-3.0,8.0) yields (-9.0,-6.0)(11.0,2.0)
Rotating (2.0,6.0)(5.0,6.0) yields (3.5,4.5)(3.5,7.5)
```

Problem 6: Shortest Paths in a Directed Graph

A *directed graph* (or *digraph*) is comprised of *vertices* (the singular of which is *vertex*) and *edges*. Each edge connects a pair of vertices, going from one to the other. In the picture below, the vertices and edges are represented by circles and arrows, respectively. Simply for the purpose of identifying the vertices, we have numbered them.

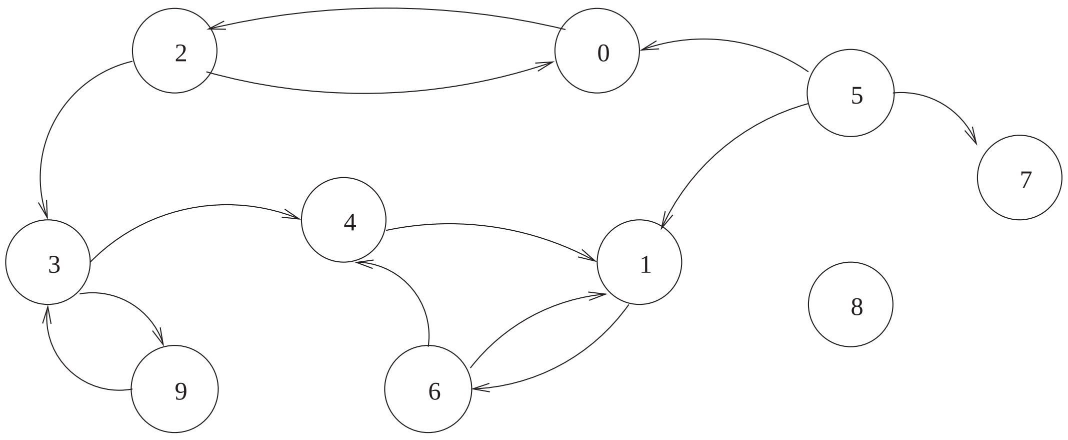

Figure 2: A 10-vertex digraph

A *path* of length m in a digraph is a sequence $\langle v_0, v_1, v_2, \ldots, v_m\rangle$ of vertices such that, for each i satisfying $0 \le i < m$, there is an edge from v_i to v_{i+1}. (Notice that the length of a path corresponds to the number of edges that are "crossed" in tracing that path.) Examples of paths in the pictured digraph are $\langle 2, 0, 2, 3, 4, 1, 6, 4, 1\rangle$ and $\langle 5\rangle$.

Suppose that u and v are vertices. There may be one or more paths from u to v, or there may be none. In the former case, the *distance* from u to v is defined to be the smallest among the lengths of those paths.

One common way of representing a digraph of n vertices is by an $n \times n$ matrix of 0's and 1's. An edge from vertex i to vertex j is indicated by a 1 appearing in the ith row and jth column of the matrix. A 0 appearing there indicates the absence of such an edge. This matrix is commonly referred to as the digraph's *adjacency matrix*.

Develop a program that, given (the adjacency matrix of) a digraph and some pairs of vertices, calculates, for each such pair, the distance from one vertex to the other.

Hint: In case u and v are the same vertex, the distance from u to v is zero. Otherwise, to find the distance from u to v, "explore" all the edges leaving u, by which you will "discover" all the vertices at distance one from u. If any of these vertices is v, its distance from u is one. Otherwise, explore all the edges leaving these vertices, by which you will discover all the vertices at distance two from u. If any of them is v, its distance from u is two. And so on and so forth. In order to avoid getting caught in an infinite loop, note that once a vertex's edges have been explored, there is no benefit in exploring them for a second time.

Input: The first line contains a positive integer n indicating the number of vertices in the digraph. On the next n lines is the digraph's adjacency matrix; each line contains a sequence of n zero's and one's, separated by spaces. (The sample input below includes the adjacency matrix for the digraph illustrated above.) On the following line is a positive integer r indicating how many pairs of vertices are to be analyzed. These pairs appear, one pair per line, on the next r lines. Each pair is given by two integers in the range $0..n-1$, separated by a space. (The vertices are assumed to be numbered from 0 to $n-1$.)

Output: For each pair of vertices given, the program is to display them and report the distance from one to the other. In case there is no path from one to the other, the program reports this fact.

```
Sample input:
-------------
10
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
1 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 1
0 1 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 1 0 0
0 1 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
8
0 6
8 4
5 6
6 5
5 4
2 3
1 1
5 9
```

```
Resultant output:
-----------------
distance from 0 to 6 is 5
no path from 8 to 4
distance from 5 to 6 is 2
no path from 6 to 5
distance from 5 to 4 is 3
distance from 2 to 3 is 1
distance from 1 to 1 is 0
distance from 5 to 9 is 4
```