ACM South American Regional

Collegiate Programming Contest

Contest Session

November 13, 1999

Buenos Aires – Argentina
Recife – Brazil
Rio de Janeiro – Brazil
Talca – Chile
Caracas – Venezuela

PROBLEM 1: Burrows Wheeler Decoder

File Names

Source File: bw.pas, bw.c, bw.cpp, bw.java
Input File: bw.in
Output File: bw.out

Statement of the Problem

The *Burrows Wheeler transform* is used in one of the most effective text compression methods. We explain the transform below by using, as an example, the input text `BANANA`.

Step 1 Let t be the size of the input text (in our example $t = 6$). We obtain a matrix M with dimensions $t \times t$. The first line of M corresponds to the input text. The i-th line, for $i = 2, \ldots, t$, corresponds to a circular left shift of $i-1$ positions with respect to the input text. For `BANANA`, we obtain the following matrix M:

```
   1 2 3 4 5 6

1  B A N A N A
2  A N A N A B
3  N A N A B A
4  A N A B A N
5  N A B A N A
6  A B A N A N
```

Step 2 We sort the lines of matrix M lexicographically to obtain a new matrix P. In our example, we get the following matrix P:

```
    1 2 3 4 5 6

1   A B A N A N
2   A N A B A N
3   A N A N A B
4   B A N A N A
5   N A B A N A
6   N A N A B A
```

Step 3 The Burrows Wheeler transform for `BANANA` is then given by the last column (i.e. column t) of matrix P and by the number of the row of P that corresponds to the input text. In our example, the last column of P is `NNBAAA` and the row of P which contains `BANANA` is the fourth one. Therefore, the Burrows Wheeler transform for `BANANA` is the pair `(NNBAAA, 4)`.

The goal of this problem is to implement the *Inverse Burrows Wheeler transform*. Given the last column of a matrix P and the number of the row of P which contains the input text, your program should obtain the input text.

Input Format

The input file may contain several instances of the problem, occurring consecutively in the input file, without any blank lines separating them. Each instance has two lines:

1. The first line contains the entries in the last column of matrix P, starting from the first row. Each entry is an uppercase letter, NOT separated by any blank spaces from its neighbour entries.
2. The second line contains the integer number corresponding to the row of P which contains the input text.

The last instance of the input file consists of the two lines

```
END
0
```

You may assume that the input text of each instance contains at most 300 letters.

Output Format

For each instance of the problem, your program should print the input text as consecutive letters in a single line. The output of each instance is separated from the next by a blank line.

Sample Input

```
NNBAAA
4
OMOEULCG
1
END
0
```

Sample Output

```
BANANA

COGUMELO
```

PROBLEM 2: Domino Game

File Names

Source File: domino.pas, domino.c, domino.cpp, domino.java
Input File: domino.in
Output File: domino.out

Statement of the Problem

Let us consider the following version of the Domino game:

- A domino piece has two sides, each of them numbered from 0 to 6.
- The game is played with 28 pieces. A piece having both sides with the same number is called a double piece.
- We will consider a two-player version of the game: each player receives a set of pieces and both sets are equal in size.
- The player who received the largest-valued double piece begins the game placing down this piece, thus starting a configuration with two extremes.
- Each player, in turn, has to place down one of his (her) pieces, always at one of the extremes of the current configuration, if and only if that piece has one of its sides matching the number at the outer side of the extreme piece. The piece is placed so that the sides with matching numbers are adjacent to each other. If the piece being placed is a double piece, it is treated as any other piece.
- Whenever a player has no piece matching one of the extremes, then he (she) passes his (her) turn.
- The winner is that player who first places down all of his pieces.

Suppose the two players are named Red and Green. Given initial sets of pieces assigned to each of Red and Green, your problem is to determine whether (i) only one of them can win, or (ii) both of them can win, or (iii) none of them can win the game. **Remember that this is not a problem of finding winning strategies. You just have to find out the winning possibility of each player**.

Input Format

The input file may contain several instances of the problem. Each instance consists of one line in the following format:

```
n  r11  r12  r21  r22  ...  rn1  rn2  g11  g12 ...  gn1  gn2
```

where:

- `n` is the size of the subsets assigned to Red and Green; you may assume that $1 \leq n \leq 14$;
- `ri1 ri2` is the i-th piece assigned to player Red;
- `gi1 gi2` is the i-th piece assigned to player Green;
- no pieces are repeated;
- at least one player is assigned a double piece;
- each of `ri1 ri2 gi1 gi2` is an integer number from 0 to 6.

An arbitrary number of blank spaces may separate a number from its neighbours in the input line of an instance. Consecutive instances are NOT separated by blank lines. The last line has $n = 0$.

Output Format

For each instance of the problem, your program should print one line containing one of the four messages:

```
Only Player Red can win
Only Player Green can win
Both Players can win
No Player can win
```

A blank line separates the outputs of two consecutive instances.

Sample Input

Sample Output

PROBLEM 3: The MTM Machine

File Names

Source File: mtm.pas, mtm.c, mtm.cpp, mtm.java
Input File: mtm.in
Output File: mtm.out

Statement of the Problem

The MTM is one of the first digital machines ever designed. The aim of the machine is to process positive integer numbers, but due to the primitive nature of the machine only some numbers are accepted for processing; such numbers are called *acceptable*. When a number is accepted by MTM, the machine outputs another number, according to the rules stated below. When a number is not accepted, the machine simply outputs `NOT ACCEPTABLE`.

A number is a non empty string of decimal digits. Given two numbers N and M, when we write NM we mean the number formed by the digits of N followed immediately by the digits of M. For example, if N is 856 and M is 112 then NM is 856112. For any number X, the *associate* of X is the number $X2X$. For example, the associate of 78 is 78278.

We say that a number X *produces* a number Y, if number X is acceptable and when given as input to the machine MTM, the number returned by the machine is Y.

The behaviour of the MTM machine is governed by the following rules:

Rule 0: A number containing the digit 0 (zero) is not acceptable.

Rule 1: Given any number X not containing a digit zero, then number $2X$ produces X. For example, 234 produces 34.

Rule 2: Given any pair of numbers X, Y, if X produces Y then $3X$ produces the associate of Y. For example, 25 produces 5 by Rule 1, so 325 produces 525.

Rule 3: No other numbers are acceptable.

Your task here is to write a program that simulates the MTM machine.

Input Format

The input file contains a set of test cases. Each test case appears in a separate line, and consists of a single positive number N, $N < 10^{32}$, to be processed by the MTM machine. The file ends with a line containing the number 0 that should not be processed.

You may assume that the largest number output by the machine has at most 1000 digits.

Output Format

For each test case, your program should write one line with the output produced by the machine if the corresponding number is acceptable; otherwise your program should write `NOT ACCEPTABLE`.

Sample Input

```
20
22
42
32
33289
0
```

Sample Output

```
NOT ACCEPTABLE
2
NOT ACCEPTABLE
NOT ACCEPTABLE
89289289289
```

PROBLEM 4: Triangular Museum

File Names

Source File: museum.pas, museum.c, museum.cpp, museum.java
Input File: museum.in
Output File: museum.out

Statement of the Problem

A museum has K^2 triangular rooms, where $(0 \leq K \leq 10)$. All rooms have the same size and there is exactly one guard in each room. The shape of the museum itself is also triangular, as exemplified in Figure 1, where a museum with 3^2 triangular rooms is shown. The names AA, BB, CC,... are guard names (a guard name is a sequence of at most 10 letters). Thus guard DD is at the top room of the initial configuration presented in Figure 1.

Each two adjacent rooms are separated by exactly one wall, which has a door that connects the rooms. Since the guards would also like to appreciate the art in the museum, they agree to exchange their positions, thus forming a new configuration, previously agreed upon. The order in which the guards move to their new positions obey a few rules. These rules, for security reasons, do not allow guards to leave their positions all at the same time. The only way they can move is by exchanging their positions with guards in adjacent rooms, one pair at a time.

The objective of this problem is to find a sequence of exchanges between guards that leads to the new configuration.

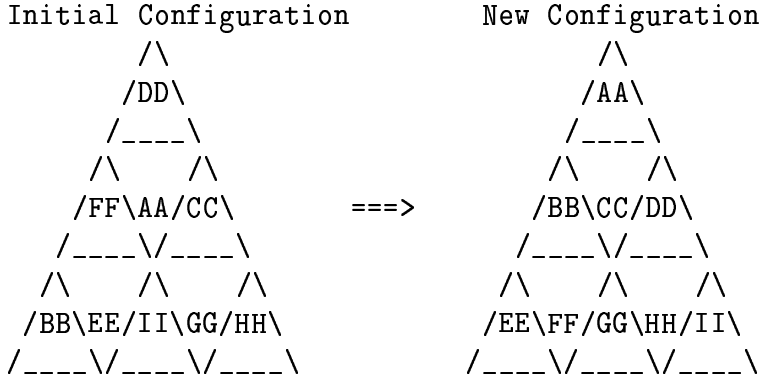

Figure 1.

Input Format

The input file may contain several instances of the problem. Each instance consists of two lines: the first contains the integer K; the second line contains the sequence of K^2 guard names in the initial configuration (from the top to bottom and left to right) followed by the

K^2 guard names in the new configuration (also from the top to bottom and left to right). There is at least one blank space between these strings of letters. The last line of the input file contains only the value 0 (zero) which should not be processed.

Instances are NOT separated by blank lines.

Output Format

For each instance in the input file, the program must write the value N in a line, where N is the number of exchanges performed between guards in adjacent rooms, followed by N lines, $N \leq 4K^3$, each with the names of two guards been exchanged; the names of the guards are separated by a blank space. Two instances are separated by a blank line.

Sample Input

```
3
DD FF AA CC BB EE II GG HH          AA BB CC DD EE FF GG HH II
2
D  A  B  C                          A  B  C  D
0
```

Sample Output

```
6
GG II
HH II
EE BB
DD AA
BB FF
DD CC

3
A B
A D
C D
```

PROBLEM 5: Numeric Puzzles Again!

File Names

Source File: npuzzle.pas, npuzzle.c, npuzzle.cpp, npuzzle.java
Input File: npuzzle.in
Output File: npuzzle.out

Statement of the Problem

A different and exciting game is invading all the toy stores around Latin America. It looks the same as a children's jigsaw, but the pieces are constructed entirely using numbers...

The pieces may have non-uniform shapes, but they all must construct a perfect $N \times N$ image. For instance, a 5×5 image may be this:

```
11223
11223
12233
11233
11133
```

Made with three pieces:

```
11
11
1
11
111

 22
 22
22
 2

 3
 3
33
33
33
```

Your problem is to write a program that solves the puzzle using the apropriate pieces.

Once the puzzle is complete, it can be rotated. It is guaranteed that at least four puzzles may be constructed with the pieces (at least *one* and its four rotations!). The pieces must NOT be rotated in order of complete the jigsaw. Each solution must use **all** the pieces.

Input Format

The input file may contain several instances of the problem. Each instance has the following lines, all consecutive in the file:

- One line with an integer giving the side length of the puzzle (at most 20).
- One line with the number of pieces (at most 9).
- Several lines describing the pieces. Each piece is made up of some combination of the same digit ($1 \ldots 9$). The pieces are left-aligned, and need not appear in any particular order.

Blank spaces may be used at the beginning of a line and within the pieces in order to define the piece's shape.

Each instance ends with a line containing only the # character. The input file ends with a line containing only the integer 0 (zero).

Output Format

You must display the *right* puzzle as output. In order to find the right puzzle, you must sum the literal values of the rows of each possible rotation of the image and return the image with the major total sum. For instance, for the puzzle

```
2 2 3
2 3 3
1 1 3
```

the sums are: $223 + 233 + 113 = 569$, $333 + 231 + 221 = 785$, $311 + 332 + 322 = 965$ and $122 + 132 + 333 = 587$. Thus the right puzzle is

```
3 1 1
3 3 2
3 2 2
```

The output of each instance must end with a blank line.

Sample Input

```
7
6
3333
33
3333
 33
```

```
 33
7   7
7   7
7777
88888
 6
666
 66
  22
222
  2
  5
 55
 55
555
5 5
#
0
```

Sample Output

```
8777333
8733333
8733323
8777323
8655222
6665552
6655555
```

PROBLEM 6: Polygon Visibility

File Names

Source File: polygon.pas, polygon.c, polygon.cpp, polygon.java
Input File: polygon.in
Output File: polygon.out

Statement of the Problem

As a building block of a new graphic tool, a visibility function has to be designed: Given two convex polygons in, it is necessary to identify the set of points **in the plane** from which the visibility of the second polygon is obstructed by the first one. More precisely, you are looking for the points x, not in the interior of the first polygon, for which there is a point y in the second polygon such that the *interior* of the line segment (x, y) contains some point of the first polygon. Your task is to write a program that implements that building block.

You may assume that the two polygons have no common points and that neither one is degenerate.

The set of points from which the visibility is obstructed can be described by a finite set of finite segments together with two infinite lines. Remember that the points in the interior of the first polygon are not included in the set of points from which the visibility is obstructed.

Input Format

The input file contains several instances. Each instance consists of the following consecutive lines:

- One line containing two positive integers N and M separated by one or more blank spaces. These are the number of vertices of the first and the second polygons. You may assume $3 \leq N, M \leq 10000$.
- N lines containing each a pair of coordinates in the XY plane, separated by one or more blank spaces. These are the vertices of the first polygon. Every coordinate is an integer number in the range 1..10000.
- M lines containing each a pair of coordinates in the XY plane, separated by one or more blank spaces. These are the vertices of the second polygon.

The vertices of each polygon appear in clockwise order. The input file ends with a line containing 0 0 and should not be processed.

Output Format

For each instance, you must write a line that identifies that instance, followed by a description of the set of points from which the visibility is obstructed. The description is formed by one of the infinite lines, the vertices that define the finite segments, and the second infinite line. Each line and each vertex must appear in a separate line of output. Each infinite line must be represented by the slope of the line, rounded to three digits to the right of the decimal point. If the line is vertical, the word VERTICAL must replace its slope value. Each vertex must be represented with its coordinates, separated by a space. To avoid ambiguity, the infinite lines and vertices must be given in counter-clockwise order.

Sample Input

```
3 3
10 20
20 20
20 10
10 10
7 5
5 7
4 3
20 19
25 15
20 10
15 15
10 20
10 10
5 15
0
```

Sample Output

```
Instance 1
VERTICAL
10 20
20 20
20 10
0.000
Instance 2
1.000
15 15
20 19
```

```
25 15
20 10
-1.000
```

PROBLEM 7: Gridsoccer

File Names

Source File: soccer.pas, soccer.c, soccer.cpp, soccer.java
Input File: soccer.in
Output File: soccer.out

Statement of the Problem

An important soccer match is about to begin between long time rival teams A and B. Team B won the toss for the ball, so a player of team B is placed exactly at the center-field position, ready for the first kick. In the defensive field of team A there is a number of players from that team in strategic positions, but no other player from team B is in that area.

At that moment, the coach for team B realizes that, if his team could move the ball really fast, the defense players of team A would have time to reach only a limited area around their original locations. Hence, if he could compute the ideal location for his players so that they could take advantage of that situation, performing the shortest sequence of kicks to reach the goal without entering the area covered by a player of team A, then the only important thing to consider would be the initial positions of team A players and their reachable areas.

Team B hires you to develop a computer program to help their coach in such a situation. The defensive field of team A is represented by a square area in the XY plane, with the center-field position at location $(0, 0)$, and the mid-field line coinciding with the X axis. This square area is partitioned into a grid with $2s$ squares in each dimension (s is given), and the goal towards which team B attacks is on the $Y = 2s$ line, occupying g squares to each side of the Y axis.

Team A players are located at grid intersections, and each one can intercept the ball if and only if it enters any square within p around its position, stopping the play (that is, each player covers an area of $(2p)^2$). No team player from team A will be at a position to cover the center-field, $(0, 0)$.

Your program should identify if there is a possibility of placing up to n players from team B (including the center field one) in grid intersections, in order to move the ball from the center field position to the goal of team A, avoiding any square of the grid reachable by a team A player. If it is not possible, notify saying: "Situation Impossible". If it is possible, say how many kicks are necessary, and identify a sequence with the smallest number of players to accomplish it. If there is more than one solution, then report the one for which the ball trajectory is the shortest.

Input Format

The input consists of a sequence of scenarios followed by a line starting with 0, which must not be processed. Each scenario consists of one line containing four integers s, g, p and n,

where

- s is the number of grid squares, in each side of the Y axis; assume $s \leq 1000$.
- g is the size of the goal, in number of grid squares in each side of the Y axis; assume $g \leq 200$.
- p is a player's reach, in number of grid squares; assume $p \leq 10$.
- n is the number of team A players in the defense field of team A; assume $n \leq 20$.

These four integers are followed by n pairs of integers defining the positions of team A players in the defense field of team A.

Output Format

The output consists of a line identifying the scenario followed by a line saying either

- Situation Impossible, or
- `Goal with at least` k `kicks: x1 y1 x2 y2 ... xk yk`, where `xi yi` indicate the position of the i-th player of team B able to kick the ball in order to accomplish the goal in k plays. Note that `x1` $= 0$ and `y1` $= 0$.

Sample Input

```
7 2 3 1   0 7
10 3 4 1  -4 9
10 3 4 2  -4 9  4 9
10 3 4 1  -3 9
0
```

Sample Output

```
Scenario 1. Situation Impossible
Scenario 2. Goal with at least 1 kicks: 0 0
Scenario 3. Goal with at least 1 kicks: 0 0
Scenario 4. Goal with at least 2 kicks: 0 0  2 0
```