ACM International Collegiate Programming Contest 2004

South American Regional

November 13th, 2004

(This problem set contains 8 problems; pages are numbered from 1 to 16)

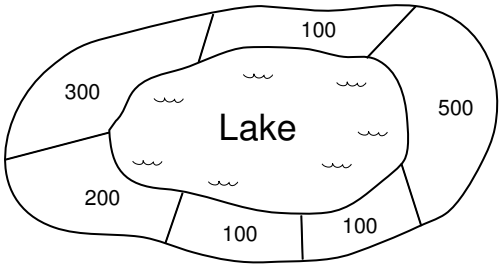
Participating countries:

- Argentina
- Brazil
- Chile
- Colombia
- Peru
- Venezuela

Problem A
Land Division Tax

Source file name: `tax.c`, `tax.cpp` *or* `tax.java`

International Concrete Projects Company (ICPC) is a construction company which specializes in building houses for the high-end market. ICPC is planning a housing development for new homes around a lake. The houses will be built in lots of different sizes, but all lots will be on the lake shore. Additionally, every lot will have exactly two neighbors in the housing development: one to the left and one to the right.



Development plan indicating the sizes of the lots (in units of area) in the new housing development.

ICPC owns the land around the lake and needs to divide it into lots according to the housing development plan. However, the County Council has a curious regulation regarding land tax, intended to discourage the creation of small lots:

1. land can only be divided using a sequence of *land divisions*;
2. a land division is an operation that divides one piece of land into two pieces of land; and
3. for each land division, a land division tax must be paid.

Denoting by A the area of the largest resulting part of the division, the value of the land division tax is $A \times F$, where F is the *division tax factor* set yearly by the County Council. Note that due to (2), in order to divide a piece of land into N lots, $N - 1$ land divisions must be performed, and therefore $N - 1$ payments must be made to the County Council.

For example, considering the figure above, if the division tax factor is 2.5 and the first land division separates the lot of 500 units of area from the other lots, the land division tax to be paid for this first division is $2.5 \times (300 + 200 + 100 + 100 + 100)$. If the next land division separates the lot of 300 units together with its neighbor lot of 100 units, from the set of the remaining lots, an additional $2.5 \times (300 + 100)$ must be paid in taxes, and so on. Note also that some land divisions are not possible, due to (2). For example, after the first land division mentioned above, it is not possible to make a land division to separate the lot of 300 units together with the lot of 200 units from the remaining three lots, because more than two parts would result from that operation.

Given the areas of all lots around the lake and the current value of the division tax factor, you must write a program to determine the smallest total land division tax that should be paid to divide the land according to the housing development plan.

Input

The input contains several test cases. The first line of a test case contains an integer N and a real F, indicating respectively the number of lots ($1 \leq N \leq 200$) and the land division tax factor (with precision of two decimal digits, $0 < F \leq 5.00$). The second line of a test case contains N integers X_i, representing the areas of contiguous lots in the development plan ($0 < X_i \leq 500$, for $1 \leq i \leq N$); furthermore, X_k is neighbour to X_{k+1} for $1 \leq k \leq N-1$, and X_N is neighbour to X_1. The end of input is indicated by $N = F = 0$.

The input must be read from the file tax.in.

Output

For each test case in the input your program must produce a single line of output, containing the minimum total land division tax, as a real number with precision of two decimal digits.

The output must be written to standard output.

Sample Input	**Output for the Sample Input**
4 1.50 2 1 4 1 6 2.50 300 100 500 100 100 200 0 0	13.50 4500.00

Problem B
Petanque

Source file name: `petanque.c, petanque.cpp` *or* `petanque.java`

Petanque, a game which uses balls (some *boules* and one *coche*), was developed in the small town of La Ciotat, near Marseilles, France. The aim of the game is simple. If playing football is kicking a ball into the opposite goal and keeping it out of your own, playing petanque is placing your boules nearer to the coche and keeping your opponent's boules away.

One simplified version of the game is played between two players, one against the other, with three boules each. The players are located in a plane and one of them starts the game by throwing the coche forward. The player who throws the coche also throws the first boule. The opposing player throws the second boule. From then on, the player who has not thrown the boule which is the nearest to the coche throws the next boule (you may assume that there will be only one boule nearest to the coche). This continues until one player has thrown all his boules. After this, the other player throws all his remaining boules. The winner is the player who has thrown the ball which in the end is nearest to the coche (again, you may assume that there will be only one boule nearest to the coche). The winning player claims as many points as there are boules thrown by him nearer the coche than any other opposing boules.

You are invited to simulate some games of Petanque. For simulation purposes, we will consider that the playing floor is a plane and the balls are adimensional (they can be considered as points). You will be given the names of the two players. The first player given will be the first to play. For each ball (coche or boules) thrown, you will be given the starting position of the ball, its direction and how many meters it will roll. You may assume that the starting position of the balls are inside the playing floor, the balls will never leave the floor and the starting position of a ball will never coincide with the current position of another ball in the playing floor. The direction of each ball is given in degrees where the east direction (x-positive) corresponds to zero degrees and the degrees increase in the counter-clockwise direction, as shown in the figure below. When a ball that was thrown knocks another ball in its way, the

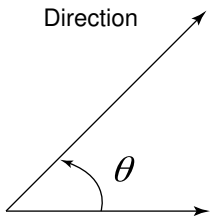


Illustration of the direction of the balls

thrown ball stops. The ball which was still starts moving in the same direction for an amount of meters equal to the quantity of meters that the thrown ball would still roll if there was no ball in its way (this amount will NEVER be zero meters). If this ball encounters another one, the procedure is repeated (here again the amount of meters still to roll will never be zero).

Input

The input consists of several test cases. The first line in the input contains an integer N specifying the number of test cases ($1 \leq N \leq 10000$). The first line of a test case contains the name of the two players separated by one space. A player name is composed of at most 20 letters from the English alphabet (from 'A' through 'Z' and 'a' through 'z'). The next *seven* lines describe the *seven* balls in the order they are thrown in the game (first the coche and then the boules thrown by the players according to the rules above). Each line contains four integers X, Y, θ and D representing respectively the starting position of a ball (-1000 $\leq X, Y \leq$ 1000), its direction ($0 \leq \theta \leq 360$) and the distance ($0 < D \leq 100$) it will roll if it does not encounter any other ball in its way.

The input must be read from the file petanque.in.

Output

For each test case in the input, your program must produce one line containing the name of the player who won the game and how many points she/he scored separated by a single space.

The output must be written to standard output.

Sample Input	**Output for the Sample Input**
2 Alex BobNelson 0 0 90 30 0 0 90 20 0 0 0 10 -5 0 0 5 0 40 270 18 0 10 90 13 -1 24 315 1 Alice Alex 0 0 90 50 0 0 90 20 0 0 180 20 -5 0 0 5 0 40 270 18 0 10 90 13 -1 24 315 1	Alex 3 Alice 1

Problem C
Long Night of Museums

Source file name: `longnight.c, longnight.cpp` *or* `longnight.java`

The city of Vienna is called the "City of Culture" because (among other things) there are more than 100 museums in the city. As a consequence, it is very difficult (and very expensive) to visit all of them no matter how long you stay in the city. Fortunately, there is a special night, called the "Long Night of Museums", when you can visit many museums with just one ticket, from 6:00 pm to 1:00 am of the next day.

Nevertheless, it is impossible to visit every museum of the city for two main reasons. First, some museums in Vienna don't get involved into this event because they close at 5:00 pm. Second, there is not enough time in 7 hours to go to every museum, watch ALL their insides (otherwise, it would be a waste of time), and then go to the others.

Given the number of museums participating in the Long Night of Museums, the time needed to watch the insides of each museum, and the time that it will take to get from each museum to the others, you have to find the best tour to visit as many museums as you can during the Long Night of Museums.

Input

The input contains several test cases. The first line of a test case contains one integer N, which indicates the number of museums participating in the event ($1 \leq N \leq 20$). Each museum has a unique identification number ranging from 1 to N. The second line of a test case contains N integers indicating the time, in minutes, needed to visit each museum, from 1 to N. Then there are N lines describing the times to go from one museum to every other. The i-th line contains N integers M_k ($1 \leq k \leq N$) representing the time, in minutes, to go from museum i to museum k. You may assume that the i-th integer in the i-th line is equal to zero. The end of input is indicated by $N = 0$

The input must be read from the file longnight.in.

Output

For each test case in the input, your program must produce one line containing the maximum number of museums that can be visited during the Long Night of Museums.

The output must be written to standard output.

Sample Input	**Output for the Sample Input**
2 500 500 0 120 200 0 2 220 220 0 30 20 0 2 150 150 0 120 200 0 0	0 1 2

Problem D
Magic Trick

Source file name: `magic.c`, `magic.cpp` *or* `magic.java`

A magician invented a new card trick and presented it in the prestigious American Conference of Magicians (ACM). The trick was so nice it received the 'Best Magic Award' at the conference. The trick requires three participants: the magician himself, a spectator and an assistant. During the trick the spectator is asked to shuffle a deck of 52 cards and pick randomly 5 cards out of the deck. The five cards are given to the assistant (without the magician seeing the cards) who looks at them and shows four of the five cards one by one to the magician. After seeing the four cards the magician magically guesses the missing fifth card!

The trick works because once the assistant has the five cards he can always choose four of them and use those to 'code' information about the fifth one. The code is based on an ordering of the cards. Cards are ordered first by their suits and then by their face value. We will use the following order:

- H < C < D < S (Hearts, Clubs, Diamonds, Spades) for suits; and
- 1 < 2 < ... < 9 < T < J < Q < K for face values, where T, J, Q and K stand for Ten, Jack, Queen and King, respectively.

Assume the spectator chose the cards JD, 8S, 7H, 8C, QH (Jack of Diamonds, 8 of Spades, 7 of Hearts, 8 of Clubs and Queen of Hearts). The strategy for the assistant is the following:

- Find a suit s which appears at least twice in the set of chosen cards (Hearts in the example). If more than one suit appears two times, choose the one with lowest order.
- Hide the card x with suit s that is at most six positions ahead in the cyclic order 1 < 2 < ... < T < J < Q < K < 1 < 2 < ... of another card y of the same suit. That is always possible since there are only thirteen cards with the same suit (in the example the assistant hides QH). If two or more cards satisfy the criteria above, choose the one with the smallest face value.
- Show y to the magician. At this point the magician knows the suit of the hidden card, and also knows that the face value of the hidden card x is at most six positions in front of the face value of y.
- With the three cards the assistant has left, he must code a number between 1 and 6. That can be done as follows. Say the three cards z_1, z_2, z_3 are in the order $z_1 < z_2 < z_3$. Each of six possible orders in which these three cards can be shown may be interpreted to convey information about a number.
  - z_1, z_2, z_3 means 1,
  - z_1, z_3, z_2 means 2,
  - z_2, z_1, z_3 means 3,
  - z_2, z_3, z_1 means 4,
  - z_3, z_1, z_2 means 5,

– z_3, z_2, z_1 means 6.

In this way, once the magician is shown the four cards one by one, he has enough information to "magically" guess the fifth one!

Your job is to develop a program that, given the four cards shown by the assistant, informs the magician which is the hidden card.

Input

The input contains several test cases. The first line in the input contains an integer N specifying the number of test cases ($1 \leq N \leq 10000$). Each test case is composed by one line, which contains the description of the four cards, separated by a space, in the order they were presented by the assistant.

The input must be read from the file magic.in.

Output

For each test case in the input your program must produce one line of output, containing the description of the hidden card.

The output must be written to standard output.

Sample Input	**Output for the Sample Input**
2 7H 8S 8C JD TC 2D 1S 5H	QH 1C

Problem E
Two-Stacks Solitaire

Source file name: `2stacks.c`, `2stacks.cpp` *or* `2stacks.java`

Card games for one player are called *Patience* in Britain and *Solitaire* in the United States. One very difficult (some say maddening) solitaire game is called Two-Stacks and has the following rules:

Tableau

The tableau consists of a stock pile, two intermediate piles and one foundation pile.

Cards

A game may use up to four complete decks, or parts of those four decks. A complete deck contains 52 cards; since all cards in a deck can be ordered, in this description we will forget about faces and suits and we will use the numbers from 1 to 52 to represent the cards in a deck.

Dealing

Once the cards to be used in the game have been chosen, they are dealt face up in the tableau, one on top of the other, forming the stock pile.

Moving cards

Cards may be moved one at a time. A card can be moved from the stock pile to one of the intermediate piles, or from one intermediate pile to the foundation pile. In a pile (stock or intermediate) only the topmost card can be moved, although all the others are visible.

Object

The goal is to have all cards used in the game in non-decreasing order, from bottom to top, constituting the foundation pile.

As you may have noticed by now, even when a solution exists, chances of winning a Two-Stacks solitaire game are very low. But your grandmother has just learned the game and loved it. She has asked you to help her to learn playing the game, by writing a program that would coach her through her first tries, showing which movements to make.

Input

The input consists of several test cases. The first line of a test case contains a single integer N ($1 \leq N \leq 208$), representing the number of cards in the game. The second line of a test case contains a sequence of N integers (between 1 and 52), separated by single blank spaces, representing the cards. The cards will be dealt in the order they appear in the input, so that the topmost card in the stock pile is the Nth card in the line. Notice that each number from

1 to 52 will appear at most 4 times in each test case. The end of input is indicated by a test case with $N = 0$.

The input must be read from the file 2stacks.in.

Output

For each test case in the input your program must produce an answer. The first line of an answer must contain a test case identifier, in the form '`#i`' where `i` starts from 1 and is incremented for every test case. Then, if it is possible to win the game, print a sequence of movements to win the game. Each movement must be described in a separate line, in the form '`push x`' or '`pop x`' where '`x`' is 1 or 2; '`push x`' moves the topmost card from the stock pile to intermediate pile `x`, and '`pop x`' moves the topmost card from intermediate pile `x` to the foundation pile. If more than one solution exists, print any one. If it is not possible to win the game, print a line with the word '`impossible`'.

The output must be written to standard output.

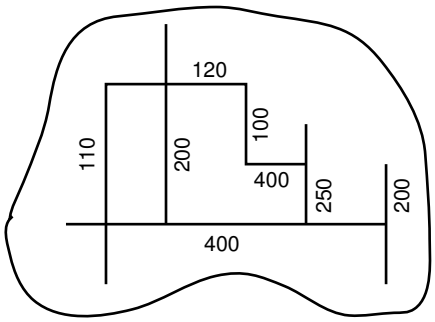
Sample Input	**Output for the Sample Input**
4 4 1 3 2 4 1 4 3 2 4 2 2 2 1 0	#1 push 1 push 2 push 1 pop 1 pop 1 push 1 pop 2 pop 1 #2 impossible #3 push 1 push 2 push 1 push 2 pop 2 pop 2 pop 1 pop 1

Problem F
Zing Zhu's Oyster Farm

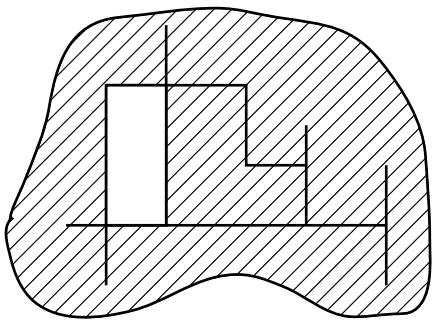
Source file name: `oyster.c, oyster.cpp` *or* `oyster.java`

Zing Zhu owns an island that is a piece of flat land. Everyday, when the tide rises, the island is flooded by sea water. After much thinking and asking advice from members of his family, Zing Zhu decided to set up an oyster farm in the island. Zing Zhu uses a sophisticated system of plastic watertight modular fences to control the areas that will be flooded and the areas that will not be flooded during the rise of the tide. The fences used by Zing Zhu are either horizontal or vertical and come in strips that have different lengths and heights. Two fences can intersect in at most one point, not necessarily in their ends.

You have been contacted by Zing Zhu to calculate, given the height the tide will reach and the position and height of all fence strips, the total area of land which will *not* be flooded during the high tide. You may assume that the widths of fence strips are so thin compared to the size



Map of fence strips installed in the farm, showing fence strip heights in centimeters



Non-flooded areas (shown in white) if the tide rises 110 centimeters

of the land that, for the purpose of calculating the total area, fence strips may be considered as having widths equal to zero.

Input

The input contains several test cases. The first line of a test case contains an integer N indicating the number of fence strips in the island ($1 \leq N \leq 2000$). Each of the next N lines contains five integers X_1, Y_1, X_2, Y_2 and H, representing respectively the start point of the strip (X_1, Y_1), the end point of the strip (X_2, Y_2) and the strip height (H). The last line of a test case contains an integer W representing the tide height. Coordinates are given in meters, heights in centimeters. Furthermore, $X_1 = X_2$ or $Y_1 = Y_2$ (but not both); $-500 \leq X_1, Y_1, X_2, Y_2 \leq 500$; and $1 \leq W, H \leq 1000$. The end of input is indicated by $N = 0$.

The input must be read from the file `oyster.in`.

Output

For each test case in the input your program must produce one line of output, containing one integer representing the total area (in m^2) of the land which will not be flooded.

The output must be written to standard output.

Sample Input	**Output for the Sample Input**
4 -20 20 20 20 200 20 20 20 -20 200 0 0 0 20 100 -10 0 20 0 200 100 4 -20 20 20 20 200 20 20 20 -20 200 0 0 0 20 100 -10 0 20 0 200 101 0	400 0

Problem G
Grandpa is Famous

Source file name: `grandpa.c`, `grandpa.cpp` *or* `grandpa.java`

The whole family was excited by the news. Everyone knew grandpa had been an extremely good bridge player for decades, but when it was announced he would be in the Guinness Book of World Records as the most successful bridge player ever, whow, that was astonishing!

The International Bridge Association (IBA) has maintained, for several years, a weekly ranking of the best players in the world. Considering that each appearance in a weekly ranking constitutes a point for the player, grandpa was nominated the best player ever because he got the highest number of points.

Having many friends who were also competing against him, grandpa is extremely curious to know which player(s) took the second place. Since the IBA rankings are now available in the internet he turned to you for help. He needs a program which, when given a list of weekly rankings, finds out which player(s) got the second place according to the number of points.

Input

The input contains several test cases. Players are identified by integers from 1 to 10000. The first line of a test case contains two integers N and M indicating respectively the number of rankings available ($2 \leq N \leq 500$) and the number of players in each ranking ($2 \leq M \leq 500$). Each of the next N lines contains the description of one weekly ranking. Each description is composed by a sequence of M integers, separated by a blank space, identifying the players who figured in that weekly ranking. You can assume that:

- in each test case there is exactly one best player and at least one second best player,
- each weekly ranking consists of M distinct player identifiers.

The end of input is indicated by $N = M = 0$.

The input must be read from the file grandpa.in.

Output

For each test case in the input your program must produce one line of output, containing the identification number of the player who is second best in number of appearances in the rankings. If there is a tie for second best, print the identification numbers of all second best players in increasing order. Each identification number produced must be followed by a blank space.

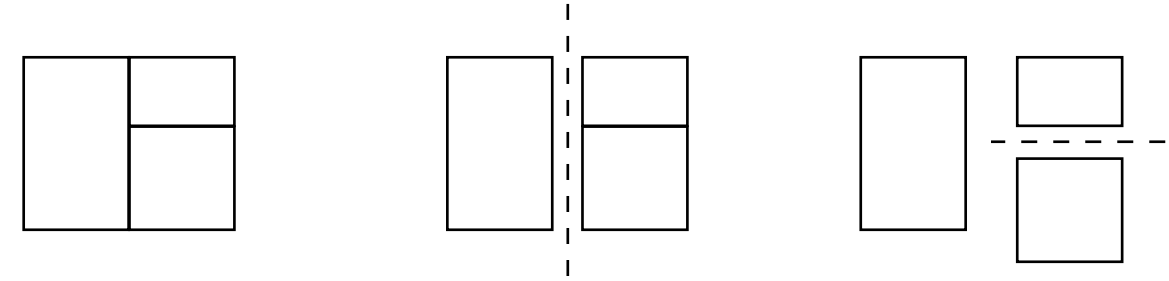
The output must be written to standard output.

Sample Input	**Output for the Sample Input**
4 5 20 33 25 32 99 32 86 99 25 10 20 99 10 33 86 19 33 74 99 32 3 6 2 34 67 36 79 93 100 38 21 76 91 85 32 23 85 31 88 1 0 0	32 33 1 2 21 23 31 32 34 36 38 67 76 79 88 91 93 100

Problem H
Cutting Edge

Source file name: `cut.c`, `cut.cpp` *or* `cut.java`

Uncle Jeff owns a glass shop, which sells glass panes for windows and picture frames. As you probably know, a glass pane can only be cut if the cut goes from edge to edge of the pane in a straight line. The figure below shows how a glass pane can be cut into three smaller glass panes.



(a) a whole glass pane (b) first cut (c) second cut

Uncle Jeff normally operates as follows. He first collects various orders for small rectangular glass panes, for windows or picture frames. He then marks the position of each small rectangular pane onto a big rectangular glass pane, such that no two marked rectangles overlap. Finally, he performs a sequence of horizontal and vertical cuts, always from edge to edge of the pane being cut, so as to produce glass panes for all the customers.

Since the last phase (the actual cutting of the big glass pane into pieces) is the most boring thing one could ever imagine, uncle Jeff is asking you for help. He wants a program which given a big rectangular glass pane and lower-left and upper-right coordinates of each marked rectangle determines the order in which the edge-to-edge cuts can be performed. This list of cuts will be fed into a machine which will do the boring cuts for him!

Input

The input contains several test cases. The first line of a test case contains an integer N indicating the number of windows and picture frames in the test ($2 \leq N \leq 2000$). Each of the next N lines contains four integers X_1, Y_1, X_2, Y_2, where (X_1, Y_1) and (X_2, X_2) represent the lower-left and upper-right coordinates marked by uncle Jeff on the big glass pane ($-5000 \leq X_1, Y_1, X_2, Y_2 \leq 5000$; $X_1 < X_2$ and $Y_1 < Y_2$). You should assume the following of each test case:

- The marked rectangles do not overlap (but may intersect on the border points) and divide the big glass pane completely into rectangular regions, so that no glass is wasted. This means that the lower-left and upper-right coordinates of the big glass pane can be inferred from the coordinates of the marked rectangles.
- It is possible to split up the big glass pane into the small marked rectangles through a sequence of edge-to-edge cuts.

The end of input is indicated by $N = 0$.

The input must be read from the file cut.in.

Output

For each test case in the input your program must produce an ordered list of cuts that must be performed to separate the big glass pane into the desired smaller panes. Each cut must appear in a different line. A cut is described by four integers X_1, Y_1, X_2, Y_2, where (X_1, Y_1) and (X_2, Y_2) specify the endpoints of the cut, with $X_1 < X_2$ and $Y_1 = Y_2$ for a horizontal cut and $X_1 = X_2$ and $Y_1 < Y_2$ for a vertical cut. As more than one ordering of cuts may be possible, your program must print the list in a particular order. If at some point more than one cut is possible, print first the cut with smaller X_1; if more than one cut is still possible, print first the one with smaller Y_1. Print a blank line after each test case list.

The output must be written to standard output.

Sample Input	**Output for the Sample Input**
3 0 0 20 30 20 0 40 20 20 20 40 30 6 1 2 2 4 2 3 3 5 1 4 2 5 2 2 3 3 3 2 4 3 3 3 4 5 0	20 0 20 30 20 20 40 20 2 2 2 5 1 4 2 4 2 3 4 3 3 2 3 3 3 3 3 5