ACM International Collegiate Programming Contest
2008 East Central Regional Contest
McMaster University
University of Cincinnati
University of Michigan – Ann Arbor
Youngstown State University
November 1, 2008

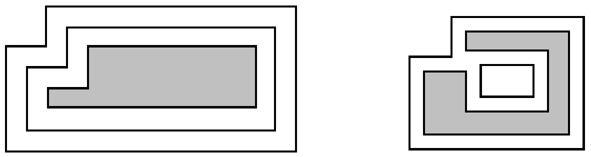
Sponsored by IBM

Rules:

1. There are **eight** questions to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases unless otherwise noted.
7. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
8. All communication with the judges will be handled by the PC^2 environment.
9. Judges' decisions are to be considered final. No cheating will be tolerated.

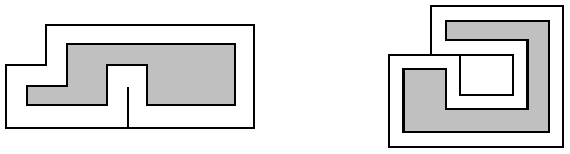
Problem A: Bordering on Madness

Bob Roberts owns a design business which creates custom artwork for various corporations. One technique that his company likes to use is to take a simple rectilinear figure (a figure where all sides meet at 90 or 270 degrees and which contains no holes) and draw one or more rectilinear borders around them. Each of these borders is drawn so that it is a set distance d away from the previously drawn border (or the original figure if it is the first border) and then the new area outlined by each border is painted a unique color. Some examples are shown below (without the coloring of the borders).



The example on the left shows a simple rectilinear figure (grey) with two borders drawn around it. The one on the right is a more complicated figure; note that the border may become disconnected.

These pieces of art can get quite large, so Bob would like a program which can draw prototypes of the finished pieces in order to judge how aesthetically pleasing they are (and how much money they will cost to build). To simplify things, Bob never starts with a figure that results in a border where 2 horizontal (or vertical) sections intersect, even at a point. This disallows such cases as those shown below:



Input

Input will consist of multiple test cases. The first line of the input file will contain a single integer indicating the number of test cases. Each test case will consist of two or more lines. The first will contain three positive integers n, m and d indicating the number of sides of the rectlinear figure, the number of borders to draw, and the distance between each border, where $n \leq 100$ and $m \leq 20$. The remaining lines will contain the n vertices of the figure, each represented by two positive integers indicating the x and y coordinates. The vertices will be listed in clockwise order starting with the vertex with the largest y value and (among those vertices) the smallest x value.

Output

For each test case, output three lines: the first will list the case number (as shown in the examples), the second will contain m integers indicating the length of each border, and the third will contain m integers indicating the additional area contributed to the artwork by each border. Both of these sets of numbers should be listed in order, starting from the border nearest the original figure. Lines two and three should be indented two spaces and labeled as shown in the examples. Separate test cases with a blank line.

Sample Input

```
2
6 2 10
20 30 100 30 100 0 0 0 0 10 20 10
10 1 7
20 50 70 50 70 0 0 0 0 30
20 30 20 10 60 10 60 40 20 40
```

Sample Output

```
Case 1:
  Perimeters: 340 420
  Areas: 3000 3800

Case 2:
  Perimeters: 380
  Areas: 2660
```

Problem B: Jack of All Trades

Jack Barter is a wheeler-dealer of the highest sort. He'll trade anything for anything, as long as he gets a good deal. Recently, he wanted to trade some red agate marbles for some goldfish. Jack's friend Amanda was willing to trade him 1 goldfish for 2 red agate marbles. But Jack did some more digging and found another friend Chuck who was willing to trade him 5 plastic shovels for 3 marbles while Amanda was willing to trade 1 goldfish for 3 plastic shovels. Jack realized that he could get a better deal going through Chuck (1.8 marbles per goldfish) than by trading his marbles directly to Amanda (2 marbles per goldfish).

Jack revels in transactions like these, but he limits the number of other people involved in a chain of transactions to 9 (otherwise things can get a bit out of hand). Normally Jack would use a little program he wrote to do all the necessary calculations to find the optimal deal, but he recently traded away his computer for a fine set of ivory-handled toothpicks. So Jack needs your help.

Input

Input will consist of multiple test cases. The first line of the file will contain an integer n indicating the number of test cases in the file. Each test case will start with a line containing two strings and a positive integer $m \leq 50$. The first string denotes the items that Jack wants, and the second string identifies the items Jack is willing to trade. After this will be m lines of the form

$$a_1 \;\; name_1 \;\; a_2 \;\; name_2$$

indicating that some friend of Jack's is willing to trade an amount a_1 of item $name_1$ for an amount a_2 of item $name_2$. (Note this does not imply the friend is also willing to trade a_2 of item $name_2$ for a_1 of item $name_1$.) The values of a_1 and a_2 will be positive and ≤ 20. No person will ever need more than $2^{31} - 1$ items to complete a successful trade.

Output

For each test case, output the phrase `Case i:` (where `i` is the case number starting at 1) followed by the best possible ratio that Jack can obtain. Output the ratio using 5 significant digits, rounded. Follow this by a single space and then the number of ways that Jack could obtain this ratio.

Sample Input

```
2
goldfish marbles 3
1 goldfish 2 marbles
5 shovels 3 marbles
1 goldfish 3 shovels
this that 4
7 this 2 that
14 this 4 that
7 this 2 theother
1 theother 1 that
```

Sample Output

```
Case 1: 1.8000 1
Case 2: 0.28571 3
```

Problem C: LCR

LCR is a simple game for three or more players. Each player starts with three chips and the object is to be the last person to have any chips. Starting with Player 1, each person rolls a set of three dice. Each die has six faces, one face with an L, one with a C, one with an R and three with a dot. For each L rolled, the player must pass a chip to the player on their left (Player 2 is considered to be to the left of Player 1); for each R rolled, the player passes a chip to the player on their right; and for each C rolled, the player puts a chip in a central pile which belongs to no player. No action is taken for any dot that is rolled. Play continues until only one player has any chips left. In addition, the following rules apply:

1. A player with no chips is not out of the game, since they may later gain chips based on other players' rolls.
2. A player with only 1 or 2 chips left only rolls 1 or 2 dice, respectively. A player with no chips left does not roll but just passes the dice to the next player.

Your job is to simulate this game given a sequence of dice rolls.

Input

Input will consist of multiple test cases. Each test case will consist of one line containing an integer n (indicating the number of players in the game) and a string (specifying the dice rolls). There will be at most 10 players in any game, and the string will consist only of the characters 'L', 'C', 'R' and '.'. In some test cases, there may be more dice rolls than are needed (i.e., some player wins the game before you use all the dice rolls). If there are not enough dice rolls left to complete a turn (for example, only two dice rolls are left for a player with 3 or more chips) then those dice rolls should be ignored. A value of $n = 0$ will indicate end of input.

Output

For each test case, output the phrase "`Game i:`" on a single line (where `i` is the case number starting at 1) followed by a description of the state of the game. This desciption will consist of n+1 lines of the form

```
Player 1:c1
Player 2:c2
...
Player n:cn
Center:ct
```

where `c1, c2` ... `cn` are the number of chips each player has at the time the simulation ended (either because some player has won or there are no more remaining dice rolls) and `ct` is the number of chips in the center pile. In addition, if some player has won, you should append the string "`(W)`" after their chip count; otherwise you should append the string "`(*)`" after the chip count of the player who is the next to roll. The only blank on any line should come before the game number or the player number. Use a single blank line to separate test cases.

Sample Input

```
3 LR.CCR.L.RLLLCLR.LL..R...CLR.
5 RL....C.L
0
```

Sample Output

```
Game 1:
Player 1:0
Player 2:0
Player 3:6(W)
Center:3

Game 2:
Player 1:1
Player 2:4
Player 3:1
Player 4:4(*)
Player 5:4
Center:1
```

Problem D: Party Party Party

Emma has just graduated high school and it is the custom for the new graduates to throw parties for themselves and invite everyone in school to attend. Naturally, Emma wishes to attend as many parties as possible. This is not such a problem on a weekday since usually there are only two or three parties in the evening. But, Saturdays are packed! Typically some parties start at 8 AM (breakfast is served) while others might end at midnight (much to the annoyance of the neighbors). Emma naturally wants to know how many parties she can attend.

Each party has a starting and stopping time, which are on the hour. These are listed via a 24-hour clock. For example, a party might start at 10 AM (10) and end at 2 PM (14). The earliest a party can start is 8 AM (8) and the latest it can end is midnight (24). In order not to be rude, Emma stays at each party at least one half hour and will consider traveling time between parties to be instantaneous. If there are times during the day when there are no parties to attend, she'll simply go home and rest.

Input

There will be multiple test cases. Each test case starts with a line containing an integer p (≤ 100) indicating the number of parties on the given day. (A value of $p = 0$ indicates end of input.) The following p lines are each of the form s e, both integers where $8 \leq s < e \leq 24$, indicating a party that starts at time s and ends at time e. Note there may be multiple parties with the same starting and ending time.

Output

For each input set output a line of the form

```
On day d Emma can attend as many as n parties.
```

where you determine the value of `n` and `d` is the number of the test case starting at 1.

Sample Input

```
8
12 13
13 14
12 13
9 10
9 10
12 13
12 14
9 11
3
14 15
14 15
14 15
0
```

Sample Output

```
On day 1 Emma can attend as many as 7 parties.
On day 2 Emma can attend as many as 2 parties.
```

Problem E: Su-Su-Sudoku

By now, everyone has played Sudoku: you're given a 9-by-9 grid of boxes which you are to fill in with the digits 1 through 9 so that 1) every row has all nine digits, 2) every column has all nine digits, and 3) all nine 3-by-3 subgrids have all nine digits. To start the game you are given a partially completed grid and are asked to fill in the remainder of the boxes. One such puzzle is shown below.

4	8	1	2	5	3	6	9	7
2	6	7	9	4	8	1		5
5	3	9	6	7	1	2		4
6	5	4	3	8	9	7	1	2
9		8	7		4	5	6	3
1	7	3	5	6	2	8	4	9
7		2	1	3	6	9	5	8
3	1	5	8	9	7	4	2	6
8	9	6	4	2	5	3	7	1

In this problem, you will be given Sudoku grids which you have nearly completed; indeed you've filled in every box except five. You are asked to complete the grid, or determine that it's impossible. (You might have already made an error!)

Input

The first line of input will contain a positive integer indicating the number of test cases to follow. Each test case will be a nearly completed Sudoku grid consisting of 9 lines, each containing 9 characters from the set of digits 0 through 9. There will be exactly five 0's in each test case, indicating the five unfilled boxes.

Output

Output for each test case should be either

```
Could not complete this grid.
```

if it is impossible to complete the grid according to the rules of the game, or the completed grid, in the form given below. (There are no blank spaces in the output.) If there is a way to complete the grid, it will be unique. Separate test cases with a blank line.

Sample Input

```
2
481253697
267948105
539671204
654389712
908704563
173562849
702136958
315897426
896425371
481253697
267948105
539671284
654289710
908704562
173562849
702136958
315897426
896425371
```

Sample Output

```
481253697
267948135
539671284
654389712
928714563
173562849
742136958
315897426
896425371

Could not complete this grid.
```

Problem F: Tanks a Lot

Imagine you have a car with a very large gas tank - large enough to hold whatever amount you need. You are traveling on a circular route on which there are a number of gas stations. The total gas in all the stations is exactly the amount it takes to travel around the circuit once. When you arrive at a gas station, you add all of that station's gas to your tank. Starting with an empty tank, it turns out there is at least one station to start, and a direction (clockwise or counter-clockwise) where you can make it around the circuit. (On the way home, you might ponder why this is the case - but trust us, it is.)

Given the distance around the circuit, the locations of the gas stations, and the number of miles your car could go using just the gas at each station, find all the stations and directions you can start at and make it around the circuit.

Input

There will be a sequence of test cases. Each test case begins with a line containing two positive integers c and s, representing the total circumference, in miles, of the circle and the total number of gas stations. Following this are s pairs of integers t and m. In each pair, t is an integer between 0 and $c-1$ measuring the clockwise location (from some arbitrary fixed point on the circle) around the circumference of one of the gas stations and m is the number of miles that can be driven using all of the gas at the station. All of the locations are distinct and the maximum value of c is 100,000. The last test case is followed by a pair of 0's.

Output

For each test case, print the test case number (in the format shown in the example below) followed by a list of pairs of values in the form `i d`, where `i` is the gas station location and `d` is either `C`, `CC`, or `CCC`, indicating that, when starting with an empty tank, it is possible to drive from location `i` around in a clockwise (`C`) direction, counterclockwise (`CC`) direction, or either direction (`CCC`), returning to location `i`. List the stations in order of increasing location.

Sample Input

```
10 4
2 3 4 3 6 1 9 3
5 5
0 1 4 1 2 1 3 1 1 1
0 0
```

Sample Output

```
Case 1: 2 C 4 CC 9 C
Case 2: 0 CCC 1 CCC 2 CCC 3 CCC 4 CCC
```

Problem G: The Worm Turns

Winston the Worm just woke up in a fresh rectangular patch of earth. The rectangular patch is divided into cells, and each cell contains either food or a rock. Winston wanders aimlessly for a while until he gets hungry; then he immediately eats the food in his cell, chooses one of the four directions (north, south, east, or west) and crawls in a straight line for as long as he can see food in the cell in front of him. If he sees a rock directly ahead of him, or sees a cell where he has already eaten the food, or sees an edge of the rectangular patch, he turns left or right and once again travels as far as he can in a straight line, eating food. He never revisits a cell. After some time he reaches a point where he can go no further so Winston stops, burps and takes a nap.

For instance, suppose Winston wakes up in the following patch of earth (X's represent stones, all other cells contain food):

	0	1	2	3	4
0					X
1					
2					
3		X	X		
4					

If Winston starts eating in row 0, column 3, he might pursue the following path (numbers represent order of visitation):

	0	1	2	3	4
0	4	3	2	1	X
1	5	18	17	16	15
2	6	19	20	21	14
3	7	X	X	22	13
4	8	9	10	11	12

In this case, he chose his path very wisely: every piece of food got eaten. Your task is to help Winston determine where he should begin eating so that his path will visit as many food cells as possible.

Input

Input will consist of multiple test cases. Each test case begins with two positive integers, m and n, defining the number of rows and columns of the patch of earth. Rows and columns are numbered starting at 0, as in the figures above. Following these is a non-negative integer r indicating the number of rocks, followed by a list of $2r$ integers denoting the row and column number of each rock. The last test case is followed by a pair of zeros. This should not be processed. The value $m \times n$ will not exceed 625.

Output

For each test case, print the test case number (beginning with 1), followed by four values:

```
amount row column direction
```

where `amount` is the maximum number of pieces of food that Winston is able to eat, `(row, column)` is the starting location of a path that enables Winston to consume this much food, and `direction` is one of `E, N, S, W`, indicating the initial direction in which Winston starts to move along this path. If there is more than one starting location, choose the one that is lexicographically least in terms of row and column numbers. If there are optimal paths with the same starting location and different starting directions, choose the first valid one in the list `E, N, S, W`. Assume there is always at least one piece of food adjacent to Winston's initial position.

Sample Input

```
5 5
3
0 4 3 1 3 2
0 0
```

Sample Output

```
Case 1: 22 0 3 W
```

Problem H: You'll be Working on the Railroad

Congratulations! Your county has just won a state grant to install a rail system between the two largest towns in the county — Acmar and Ibmar. This rail system will be installed in sections, each section connecting two different towns in the county, with the first section starting at Acmar and the last ending at Ibmar. The provisions of the grant specify that the state will pay for the two largest sections of the rail system, and the county will pay for the rest (if the rail system consists of only two sections, the state will pay for just the larger section; if the rail system consists of only one section, the state will pay nothing). The state is no fool and will only consider simple paths; that is, paths where you visit a town no more than once. It is your job, as a recently elected county manager, to determine how to build the rail system so that the county pays as little as possible. You have at your disposal estimates for the cost of connecting various pairs of cities in the county, but you're short one very important requirement — the brains to solve this problem. Fortunately, the lackeys in the computing services division will come up with something.

Input

Input will contain multiple test cases. Each case will start with a line containing a single positive integer $n \leq 50$, indicating the number of railway section estimates. (There may not be estimates for tracks between all pairs of towns.) Following this will be n lines each containing one estimate. Each estimate will consist of three integers s e c, where s and e are the starting and ending towns and c is the cost estimate between them. (Acmar will always be town 0 and Ibmar will always be town 1. The remaining towns will be numbered using consecutive numbers.) The costs will be symmetric, i.e., the cost to build a railway section from town s to town e is the same as the cost to go from town e to town s, and costs will always be positive and no greater than 1000. It will always be possible to somehow travel from Acmar to Ibmar by rail using these sections. A value of $n = 0$ will signal the end of input.

Output

For each test case, output a single line of the form

```
c1 c2 ... cm cost
```

where each `ci` is a city on the cheapest path and `cost` is the cost to the county (note `c1` will always be `0` and `cm` will always be `1` and `ci` and `ci+1` are connected on the path). In case of a tie, print the path with the shortest number of sections; if there is still a tie, pick the path that comes first lexicographically.

Sample Input

```
7
0 2 10
0 3 6
2 4 5
3 4 3
3 5 4
4 1 7
5 1 8
0
```

Sample Output

```
0 3 4 1 3
```