ACM International Collegiate Programming Contest
2011 East Central Regional Contest
Grand Valley State University
University of Cincinnati
University of Windsor
Youngstown State University
October 22, 2011

Sponsored by IBM

Rules:

1. There are **nine** problems 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. No whitespace should appear in the output except between printed fields .
4. All whitespace, either in input or output, will consist of exactly one blank character.
5. The allowed programming languages are C, C++ and Java.
6. All programs will be re-compiled prior to testing with the judges' data.
7. 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).
8. The input to all problems will consist of multiple test cases.
9. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
10. All communication with the judges will be handled by the PC^2 environment.
11. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: The Agency

Following in the footsteps of a number of flight searching startups you want to create the first inter-planetary travel website. Your first problem is to quickly find the cheapest way to travel between two planets. You have an advantage over your competitors because you have realized that all the planets and the flights between them have a special structure. Each planet is represented by a string of N bits and there is a flight between two planets if their N-bit strings differ in exactly one position.

The cost of a flight is the cost of landing on the destination planet. If the ith character in a planet's string is a 1 then the ith tax must be paid to land. The cost of landing on a planet is the sum of the applicable taxes.

Given the starting planet, ending planet, and cost of the ith tax compute the cheapest set of flights to get from the starting planet to the ending planet.

Input

Input for each test case will consist of two lines. The first line will have N ($1 \leq N \leq 1{,}000$), the number of bits representing a planet; S, a string of N zeroes and ones representing the starting planet; and E, a string representing the ending planet in the same format. The second line will contain N integers the ith of which is the cost of the ith tax. All costs will be between 1 and 1,000,000. The input will be terminated by a line with a single 0.

Output

For each test case output one number, the minimum cost to get from the starting planet to the ending planet, using the format given below.

Sample Input

```
3 110 011
3 1 2
5 00000 11111
1 2 3 4 5
4 1111 1000
100 1 1 1
30 000000000000000000000000000000 111111111111111111111111111111
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
0
```

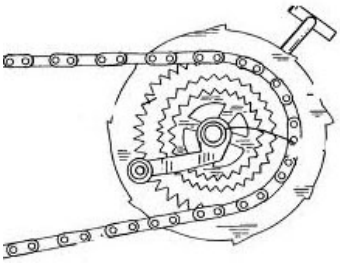
Sample Output

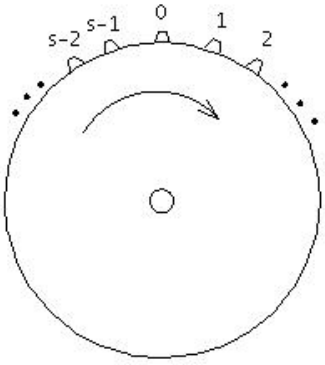
```
Case 1: 4
Case 2: 35
Case 3: 106
Case 4: 4960
```

Problem B: Chain of Fools

Many of you have heard the story of Turing's bicycle: Seems the sprocket on the crank of the bicycle had a broken prong. Also his chain had one link that was bent. When the bent link on the chain came to hook up with the broken prong, the chain would fall off and Turing would stop and put the chain back on. But Turing, being who he was, could predict just when this was going to happen — meaning he would know how many pedal strokes it would be — and so would hop off his bike just before it happened and gently move the pedals by hand as the undesired coupling passed. Then he'd be merrily (we imagine) on his way. (A picture of the sprocket-chain set up is shown below.)

Your job here is to calculate the number of revolutions required in such a situation as Turing's: You'll be given the number of prongs on the front sprocket, the number of links on the chain, the location of the broken prong and the location of the bent link in the chain. The top prong is at location 0, then the next one forward on the sprocket is location 1 and so on until prong numbered $s - 1$. (See the diagram. Notice that prong $s - 1$ is the next prong that moves to the top of the sprocket as Turing pedals.) Location of the links is similar: The link at the top of the sprocket is location 0 and so on forward until $c - 1$. The chain falls off when broken prong and bent link are both at location 0.





Input

Input for each test case will be one line of the form $s\ c\ p\ l$, where s is the number of prongs on the front sprocket ($1 < s < 100$) , c is the number of links in the chain ($200 > c > s$), p is the initial position of the broken prong, and l is the initial position of the bent link. The line 0 0 0 0 will follow the last line of input.

Broken prong and bent link will never both start at position 0.

Output

For each test case output a single one line as follows:

`Case` n: $r\ m/s$

if it requires $r\ m/s$ revolutions to first fail, or

`Case` n: Never

if this can never happen.

Note that the denominator of the fraction will always be the number of prongs on the sprocket; the fraction will not necessarily be in lowest terms. Always print the values of r and m, even if 0.

Sample Input

```
40 71 32 23
20 40 4 24
40 71 8 33
20 40 3 17
0 0 0 0
```

Sample Output

```
Case 1: 1 8/40
Case 2: 0 16/20
Case 3: 25 32/40
Case 4: Never
```

Problem C: Condorcet Winners

A *Condorcet winner* of an election is a candidate who would beat any of the other candidates in a one-on-one contest. Determining a Condorcet winner can only be done when voters specify a ballot listing all of the candidates in their order of preference (we will call such a ballot a *preference list*). For example, assume we have 3 candidates — A, B and C — and three voters whose preference lists are ABC, BAC, CBA. Here, B is the Condorcet winner, beating A in 2 of the three ballots (ballots 2 and 3) and beating C in 2 of the three ballots (1 and 2).

The *Condorcet voting system* looks for the Condorcet winner and declares that person the winner of the election. Note that if we were only considering first place votes in the example above (as we do in most elections in the US and Canada), then there would be a tie for first place. There can be at most only one Condorcet winner, but there is one small drawback in the Condorcet system — it is possible that there may be no Condorcet winner.

Input

Input for each test case will start with a single line containing two positive integers b c, where b indicates the number of ballots and c indicates the number of candidates. Candidates are considered numbered $0, \ldots, c-1$. Following this first line will be b lines containing c values each. Each of these lines represents one ballot and will contain the values $0, \ldots, c-1$ in some permuted order. Values of b and c will lie in the ranges $1 \ldots 500$ and $1 \ldots 2500$, respectively, and the line 0 0 will follow the last test case.

Output

For each test case output a single line containing either the candidate number of the Condorcet winner, or the phrase `No Condorcet winner` using the format given.

Sample Input

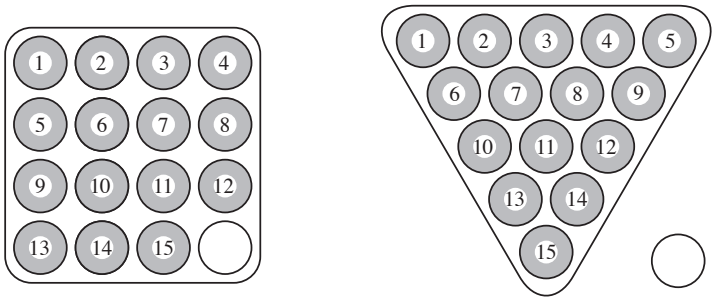
```
3 3
0 1 2
1 0 2
2 1 0
3 3
0 1 2
1 2 0
2 0 1
0 0
```

Sample Output

```
Case 1: 1
Case 2: No Condorcet winner
```

Problem D: Everyone out of the Pool

When you rent a table at a pool hall, the proprietor gives you a 4-by-4 tray of 16 balls, as shown in Figure (a) below. One of these balls, called the "cue ball", is white, and the remaining 15 are numbered 1 through 15. At the beginning of a game, the numbered balls are racked up in a triangle (without the cue ball), as shown in Figure (b).



(a) (b)

Now imagine other pool-like games where you have a cue ball and x numbered balls. You'd like to be able to rack up the x numbered balls in a triangle, and have all $x+1$ balls perfectly fill a square m-by-m tray. For what values of x is this possible? In this problem you'll be given an lower bound a and upper bound b, and asked how many numbers within this range have the above property.

Input

Input for each test case will one line containin two integers a b, where $0 < a < b \leq 10^9$. The line 0 0 will follow the last test case.

Output

For each test case one line of output as follows:

`Case` n: k

if there are k integers x such that $a < x + 1 < b$, x balls can be racked up in a triangle, and $x + 1$ balls fill a square tray.

Sample Input

```
15 17
14 16
1 20
0 0
```

Sample Output

```
Case 1: 1
Case 2: 0
Case 3: 2
```

Problem E: The Banzhaf Buzz-Off

A young researcher named George Lurdan has just been promoted to the board of trustees for Amalgamated Artichokes. Each member of the board has a specified number of votes (or *w*eights) assigned to him or her which can be used when voting on various issues that come before the board. Needless to say, the higher your weight, the more power you have on the board, but exactly how much power you have is a difficult question. It depends not only on the distribution of the weights, but also on what percentage of the votes are needed to pass any resolution (known as the *q*uota). For example, the current board has 5 members whose weights are 20, 11, 10, 8 and 1, where George has the 1. Whenever a simple majority of votes are needed (i.e., when the quota is 26) George has very little power. But when the vote has to be unanimous (quota = 50), George has just as much power as anyone else. George would like to know how much power he has depending on what the quota is; he figures that if his power is zero, then there's no point in going to the meetings—he can buzz off and spend more time on artichoke research.

After doing some reading, George discovered the Banzhaf Power Index (BPI). The BPI measures how often each board member is a critical voter in a winning coalition. A winning coalition is a group of board members whose total weights are greater than or equal to the quota (i.e., they can pass a resolution). A *c*ritical voter in a winning coalition is any member whose departure results in a non-winning coalition. For example, if the quota is 26, then one possible winning coalition would consist of 20, 10 and 1 (George). Here, each of the first two members of the coalition are critical (since if they leave the resulting coalition has only 11 or 21 votes), while George is not critical. In the winning coalition 20, 11, 10, 8, 1, no one is critical when the quota is 26, but everyone is when the quota is 50. The BPI for any member is just the total number of winning coalitions in which that member is critical (NOTE: in reality, the BPI is actually double this figure, but that's not important for us here). For example, when the quota is 26, the BPIs for the members turn out to be 12, 4, 4, 4 and 0, i.e., the board member with weight 20 is a critical voter in 12 different winning coalitions, the board member with weight 11 is a critical voter in 4 different winning coalitions, etc. As expected, George has no power in this case. If the quota is raised to 42, then the BPIs are 3, 3, 3, 1 and 1. In this case George has just as much power as the member with 8 votes!

Since the number of members on the board can vary from 1 to 60 , and board members' weights can change over time, George would like a general program to determine his BPI power.

Input

Input for each test case will consist of two lines: the first line will contain two integers n, q, where n indicates the number of distinct weight values ($1 \le n \le 60$) and q is the quota. The second line will contain n pairs of positive integers $w_1\ m_1\ w_2\ m_2\ \cdots\ w_n\ m_n$ where w_i is a weight value and m_i is the number of board members with that weight. The total number of votes $V = w_1m_1 + w_2m_2 + \cdots + w_nm_n$ will be in the range $1 \le V \le 60$. The quota will satisfy the condition $V/2 < q \le V$, and $w_i \ne w_j$ when $i \ne j$. A line containing "`0 0`" will signal the end of input.

Output

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

`Case` n: $b_1\ b_2\ \ldots\ b_n$

where b_i is the Banzhaf Power Index for any member with weight w_i. Separate the BPIs with a single blank.

Sample Input

```
5 26
20 1 11 1 10 1 8 1 1 1
5 42
20 1 11 1 10 1 8 1 1 1
5 50
20 1 11 1 10 1 8 1 1 1
1 31
1 60
0 0
```

Sample Output

```
Case 1: 12 4 4 4 0
Case 2: 3 3 3 1 1
Case 3: 1 1 1 1 1
Case 4: 59132290782430712
```

Problem F: GPS I Love You

Thomas T. Garmin got a GPS for his birthday last year, and he loved it! Unfortunately, sometimes Tom wanted to take a scenic route rather than the shortest one as suggested by the GPS. He did a little reading in the manual and found that he could override the default path-finding algorithm by specifying roads which the GPS system would be forced to use when determining a route. After some experimentation, Tom discovered that often, forcing a single road sufficed to get the desired route. However, for some windier routes, Tom needed to force more roads. Eventually Tom began to worry that he wasted too much time picking roads to force before each trip. Now, instead of enjoying the wonders of his GPS, he spends his drives agonizing over the following question: could he have gotten his GPS to pick the scenic route using fewer forced roads?

Can you save this love affair, or are Tom and his GPS doomed to walk separate paths?

Input

Input for each test case will consist of a number of lines. The first line will contain a single integer $n < 100$ indicating the number of endpoints for the roads, numbered 0 to $n-1$. There will then follow n lines each containing n non-negative integers. If the j^{th} value in row i is positive, it indicates the length of a road from endpoint i to endpoint j; if the value is 0, it indicates that there is no road between those two endpoints. Following these lines will be a line of the form $m\ p_1\ p_2\ p_3\ \ldots\ p_m$ specifying the scenic route that Tom wants – the route contains $m-1$ roads and goes between endpoints p_1 and p_m, visiting endpoints p_2, p_3, etc., in that order. The last test case will be followed by a line containing 0.

Note that when Tom specifies his forced roads to his GPS, he specifies both their direction and order. All the routes are simple paths and all roads lengths are ≤ 100.

Output

For each test case one line of output as follows:

`Case` n: k

where k is the smallest number of roads Tom must force such that the GPS will choose the specified route. You should assume that if there are multiple shortest paths, the GPS always selects the most scenic of these. Therefore, if Tom's route is among the shortest to use a given set of forced roads, it will be picked by the GPS.

Sample Input

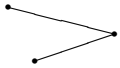
```
4
0 4 0 2
4 0 2 0
0 2 0 2
2 0 2 0
4 0 3 2 1
4
0 4 0 1
4 0 1 0
0 1 0 1
1 0 1 0
4 0 3 2 1
0
```

Sample Output

```
Case 1: 1
```

Problem G: Have You Driven a Fjord Lately?

As most of us know, the western Scandinavian coastline contains many small inlets from the sea known as *fjords*. Fjords have very steep sides, and make travel along the coast somewhat tedious (though breathtaking) as the roads must curve back and forth around them. The Fjord Accelerated Scandinavian Traffic Commission (FAST) has decided to solve this problem by putting in a series of bridges across the fjords to cut down on the distances which must be traveled. To save costs, FAST is using pre-constructed bridge units of length 1 meter each, but due to funding restrictions, the total length of bridge that they can build is limited. Therefore, they would like to determine the optimal locations to install bridges that would save the greatest length of road. Fjor example, if a bridge of length 10 meters is built that cuts off 30 meters of old road, a savings of 20 meters is realized. To simplify the determination of where to locate the bridges, FAST has decided to model each fjord as two line segments connecting three points as shown in the figure below.



All the angles making fjords are less than 180°, of course. Furthermore, for safety reasons each bridge can span at most one fjord.

Input

Input for each test case will consist of two lines. The first line contains two positive integers n and m indicating the number of fjords and the maximum length (in meters) of bridge that can be built. The next line will contain $2n + 1$ pairs of integer coordinates for the fjords, where the last coordinate for fjord i serves as the first coordinate for fjord $i + 1$. All coordinates are given in units of meters and will be between -300000 and 300000. The maximum values for n and m are 50 and 3000, respectively. Input will end with the line 0 0.

Output

For each test case output a single line containing the case number followed by the length of the bridge used and the total savings for the optimal placement of bridges, using the format shown below. All values should be in meters and round the latter number to the nearest hundredth.

Sample Input

```
2 6
0 0 4 2 0 4 2 6 0 8
2 6
0 0 4 2 0 4 8 6 0 8
2 10
0 0 4 2 0 4 8 6 0 8
0 0
```

Sample Output

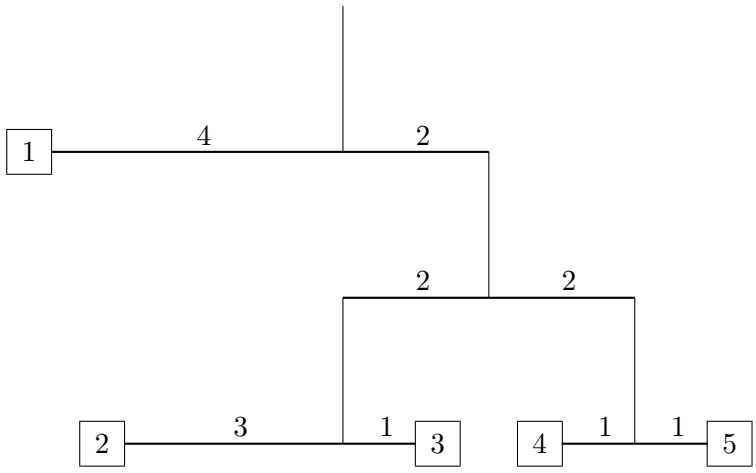
```
Case 1: 6 meters used saving 5.77 meters
Case 2: 6 meters used saving 14.96 meters
Case 3: 8 meters used saving 17.44 meters
```

Problem H: Mobile

You've probably seen mobiles suspended from the ceilings of museums or airports. We'll restrict ourselves to the type suspended from the ceiling by a single wire that is attached to a pivot point on an arm (also made of wire). At each end of the arm is either another wire suspending yet another arm, or a weight (usually in the form of some design). Below is one example, made by Alexander Calder, the best-known mobile artist.

Some mobiles are simple and some are quite complex. Besides the artistry, these must balance. Recall that from a pivot point distance d_L from the left and d_R from the right, an arm will balance if the product of the weight at the left end and d_L is equal to the product of the weight at the right end and d_R. (We ignore the weight of the arm and the wires suspending the arms.)

For example, consider the mobile drawn below. If weight 1 weighs 8 units, then weights 2, 3, 4, and 5 must weigh 2, 6, 4, and 4 units respectively. In fact, if you know the structure of the mobile, that is, the arrangement of arms and where the pivot points are on each arm, and the value of one weight, you can determine the values of all the weights. That is your problem here – almost. It seems you only have weights that are integer valued. So, you'll be given the desired minimum value of one weight and determine the value of the other weights, so that those values will also be integers. Thus, it's possible that the specified minimum valued weight must be raised a little bit to accomplish this.



Input

Input for each test case will start with a line containing the positive integer n, indicating the number of arms in the mobile. These arms are numbered 1 through n. The next n lines will describe the arms,

in order $1, 2, \ldots, n$, and will be in the form

d_L d_R $type_L$ $type_R$ n_L n_R

where d_L and d_R are integers ≤ 20 giving the distances from the pivot point to the left end and right end of the arm, $type_L$ and $type_R$ are each either `W` or `A`, indicating that a weight or arm hangs from the left or right ends, and n_L and n_R are the index numbers of the weight or arm on the left and right. The indices for the weights will start at 1 and be consecutive. The mobile will not have an arm that is hanging further down than 6 arms from the top. In our example above the lowest arm is 3 arms from the top.

Following the description of the arms is a line of the form m w, indicating that weight m weighs at least w, where $1 \leq w \leq 20$.

A line containing a 0 follows the last test case.

Output

For each test case output one line giving the minimum total weight of the mobile if weight m is at least w. Use the format given in the Sample Output. You may assume all output values will be less than 10^9.

Sample Input

```
4
3 1 W W 2 3
4 2 W A 1 3
2 2 A A 1 4
1 1 W W 4 5
1 8
4
2 2 A W 2 5
3 1 W A 4 3
4 1 W A 3 4
2 1 W W 1 2
3 20
0
```

Sample Output

```
Case 1: 24
Case 2: 280
```

Problem I: Wally World

Two star-crossed lovers want to meet. The two lovers are standing at distinct points in the plane (but then again, aren't we all?). They can travel freely except that there is a single wall which cannot be crossed. The wall is a line segment which is parallel to either the x or y axis. Each lover can move 1 unit in 1 second. How long will it take them to be together if they both choose the best path?

Input

Input for each test case will consist of two lines each containing four integers. The first two integers will specify the x and y coordinates of the first lover; the next two integers will specify the x and y coordinates of the second lover. The next four integers will specify the start and end points of the wall. Furthermore, in all cases both lovers will not be on the (infinite) line containing the wall — that is, the wall extended in both directions. All coordinates will be positive and less than or equal to 10000 and neither lover will start on the wall. The input will be terminated by a line containing four zeroes.

Output

For each test case, output the minimum time in seconds for the two lovers to meet. Print the answer to exactly 3 decimal places, using the output format shown in the example.

Sample Input

```
5 2 7 2
1 1 1 100
1 2 3 2
2 1 2 100
0 0 0 0
```

Sample Output

```
Case 1: 1.000
Case 2: 1.414
```