ACM ICPC Bolivia Programming Contest 2010

Cuarto On-Line

Set de Problemas

Domingo 30 de Mayo del 2010

- A - Comparison Expressions
- B - Esperanto Lessons
- C - Karaoke
- D - Playing with Marbles
- E - Publish of Perish
- F - The Sweeper in Cleanland
- G - Worms

PROBLEM A

COMPARISON EXPRESSIONS

Automated Compilers Manufacture, is developing a new, inteligent compiler. One of the problems its scientists are facing is how to determine whether two expressions are equivalent. An expression consists of one or more alphabetic lowercase letters, representing *variables*, separated by addition and multiplication signs, respectively + and *, totally parenthesized. Recursively, an expression is either a variable, or a string of the form (E1 + E2) or (E1 * E2), where E1 and E2 are both expressions. Examples of expressions: a, (a+b), ((a*b)*b). Examples of non-expressions: ab, a*(b+c).

Two expressions are *equivalent* if, for every integer assignment to their variables, both expressions assume the same value. For example, ((a+b)*c) and ((b*c)+(c*a)) are equivalent, whereas (a+(b*c)) and ((a+b)*c) are not equivalent. Your task is to write a program that decides whether two given expressions are equivalent.

Input

The input contains several test cases, each case consists of one line, containing two expressions separated by precisely one space. The variables of the expressions are represented by lowercase alphabetic letters. The number of occurrences of variables in an expression is at most 20 (for example, ((a+b)*a) has three occurrences of variables). Thus, an expressions consists of at most 77 characters, including variables, parentheses, plus and multiplication signs. The end of the input is indicated by a line containing only one zero.

Output

For each test case in the input, your program must print a single line, containing one single character, a Y if the two expressions are equivalent, an N otherwise.

Sample Input

```
(a+(b*c)) ((a+b)*c)
((a+b)*c) ((c*b)+(a*c))
0
```

Sample Output

```
N
Y
```

Problem B

Esperanto Lessons

In a Latin American high school the directors had decided to implement esperanto lessons, because they have observed many of their students started studying the universal language on their own. Because of this, all kids have really different levels before the courses are implemented. The directors decided to implement two levels: basic and advanced. Because of bureaucracy, they cannot put students of different divisions in the same English course. Also, to be fair, the basic and advanced levels have to be equal among all divisions of the school.

Therefore, each division will be partitioned in two subgroups, one for the basic level and one for the advanced level (note that it is possible that a division does not contain any students in one of the levels). To determine the levels, an Esperanto test has been taken by all students of all division, each getting a grade between 0 and 1000, inclusive. To accomplish their mentioned goals, the directors have decided that all students with a score greater than or equal to T will be assigned the advanced level and all students with a score less than T will be assigned the basic level.

However, they cannot decide on the best value of T. They decided to choose the value that better splits all divisions. For this, they have come up with a metric: They want the value of T that minimizes the accumulated difference. That is, the sum of the difference of the number of students in the two groups (basic and advanced) within each division.

Input

Each test case will be given in several lines. The first line contains a single integer N ($1 <= N <= 10^6$), the number of divisions in the school. 2N lines follow, with each division being described in 2 consecutive lines. The first line of each group of two contain a single integer K ($1 <= K <= 10^6$) the number of people on the division. The second line contains K integers separated by single spaces with the scores of each of the students in the division. The scores will all be between 0 and 1000, inclusive. Total number of students within each test case (that is, the sum of the values of all K lines) will not be greater than 10^6.

Output

For each test case, output a single line with a single integer representing the value of the accumulated difference if T is chosen optimally.

Sample Input

```
2
2
1 2
2
3 4
2
2
1 4
2
2 3
3
4
1 10 100 1000
```

3
5 55 555
5
4 16 64 256 1000
1
4
500 500 500 500
-1

Sample Output

2
0
2
4

PROBLEM C

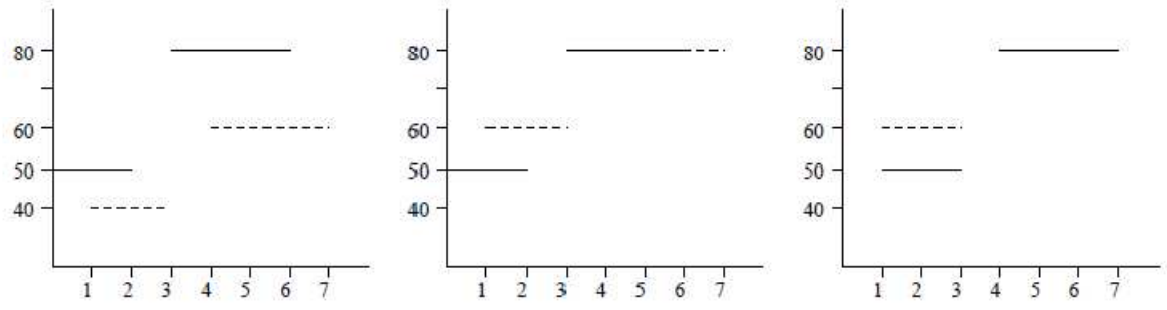
KARAOKE

"But I sing awfully!" - Gonzo kept saying. "Come on! Nobody cares, it just for fun!" – was the common reply of his friends. "And we'll give you all our support!". But Gonzo was not convinced. Although he was an accomplished shower singer, he didn't like to do it in public. He had played karaoke games before and his problems were always the same: he could pronounce the words and keep a note very well, but typically it was not the right one, and not at the correct time. That's why in these games his scores were always poor.

But there was something in his pocket to save the day. He had a friend who just came from a programming contest and had given him a pendrive with the solution: A Corrector for Music (ACM), a program that can adjust the notes that the game detects, so that he could achieve the a better score. Gonzo was relieved. For once he wouldn't be the worst singer of the group. Now he had just to use the program properly. The karaoke game displays a screen with several horizontal bars, representing words of the song, at different heights over a time axis. The y-coordinate of each bar indicates the *pitch* of that word. This is, the note that the game expects to detect from the singer. The game receives the voice of the singer from the microphone, and detects the pitch and the time interval where the note was detected. By comparison with the original song, the game displays a new set of horizontal bars in the same graph of the originals, and calculates the score according to the following rules:

- The singer starts the game with 0 points.
- If the song pitch and the singer pitch coincides, the singer wins 100 points by each second that the coincidence continues.
- If the song pitch and the singer pitch differ, the singer loses the absolute difference between the pitches each second that such difference exists.
- If there is no song pitch, but the singer sings; or inversely, if there is a song pitch, but the singer doesn't make a sound, the singer loses 100 points each second that this happens.
- While there is no song pitch, and no input from the microphone, the score remains the same.

ACM modifies the pitch of all the singer words by adding an integer to all of them. There is no possibility of modifying just an specific part of the singer's notes. The pitch is always a positive number. If by any chance a pitch number drops to zero or less, the game interprets that no sound has been received. The program can also delay the time when the song begins or the time when the singer's voice begins to be read, by adding an integer number of seconds in order to better adjust the intervals. The figure shows an example with the song words (solid lines) and the singer's words (dashed lines).



(a) Original. Score = −450 (b) Singer Pitch +20. Score = −210 (c) Song delayed 1s. Score = 280

You will receive the details of the song that is stored in the game, and the words detected by the microphone. With that information you must determine the maximum possible score that can be obtained by adjusting the pitch of the singer's words, and delaying the song or the voice.

Input

The input contains several test cases. The first line of a test case contains one integer N indicating the number of words that are included in the song (1<=N<=200). Each of the next N lines contain six integers s_{i1}, f_{i1}, p_{i1}, s_{i2}, f_{i2} and p_{i2}. The first three of those integers represent the continuous interval of time $[s_{i1}, f_{i1})$, where the word i must be sung with pitch pi1. The following three integers represent the continuous interval $[s_{i2}, f_{i2})$, where the word i of the singer is detected with pitch p_{i2}. For all cases, $0 < s_{i1} < f_{i1}$, $0 < s_{i2} < f_{i2}$, $20 < p_{i1}, p_{i2} < 120$. The end of input is indicated by a line containing only one zero.

Output

For each test case in the input, your program must print a single line, containing the maximum score that can be achieved in the game by using the ACM program.

Sample Input

```
1
1 4 60 6 9 70
1
1 4 60 6 8 70
2
0 2 50 1 3 40
3 6 80 4 7 60
0
```

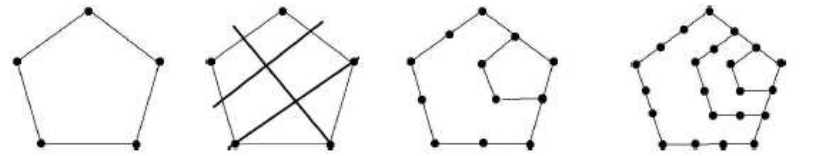
Sample Output

```
300
100
280
```

PROBLEM D

PLAYING WITH MARBLES

Pablo was assigned in his class to construct pentagons inside pentagons with marbles but he doesn't know how many marbles he will need. He knows that for one pentagon he needs 5 marbles



The only way he knows to insert a second pentagon is putting a marble in the middle of each segment and drawing three lines as shown. He puts a marble in the intersecting lines and removes them. To insert a third pentagon inside he first divides all segments in two including the ones that are not needed, and repeats the procedure. Drawing a second pentagon will require 12 marbles. A third pentagon will require 22 marbles. Given the information of how many pentagons will be created, write a program to calculate the number of marbles needed.

Input

The input contains several test cases. Each test case contains one integer N indicating the number of pentagons to create ($1<=N<=10^3$). The end of input is indicated by a line containing only one zero.

Output

For each test case in the input, your program must print a single line, containing one single integer, the number of marbles required.

Sample Input

```
1
2
3
0
```

Sample Output

```
5
12
22
```

PROBLEM E

PUBLISH OF PERISH

"Publish or perish" is the academic life's fundamental motto. It refers to the fact that publishing your work frequently is the only way to guarantee access to research funds, bright students and career advances. But publishing is not enough. It is necessary that your work is *referenced* (or *cited*). That is, your papers must be mentioned as source of information in other people's publications, to attest the quality and relevance of your research. The more citations a paper receives from other authors, the more it is considered influential.

In 2005 Jorge E. Hirsch, a physicist at the University of California at San Diego, proposed a way to evaluate the scientific impact of a researcher, based on the citations his or her papers have received. The *h-index*, as Hirsch's proposal became known, is a number based on the set of a researcher's most cited papers. It is defined in Hirsch's own words as: A scientist has index h if h of his N_p papers have at least h citations each, and the other (N_p − h) papers have at most h citations each.

Albert Einstein, for example, published 319 papers in scientific journals and has an h-index equal to 46. It means 46 of his papers have received 46 or more citations each, and all of his remaining 273 papers have 46 citations or less each. Given the information of how many citations each paper from a given researcher has received, write a program to calculate that researcher's h-index.

Input

The input contains several test cases. The first line of a test case contains one integer N indicating the number of papers a researcher has published ($1<=N<=10^3$). The second line contains a list of N integers M_i, separated by one space, representing the number of citations each of the N papers from that author has received ($0<=M_i<=10^3$, for $1<= i <= N$). The end of input is indicated by a line containing only one zero.

Output

For each test case in the input, your program must print a single line, containing one single integer, the h-index for the given list of citations.

Sample Input

```
4
1003 1 200 2
10
1 1 1 0 1 1 0 1 1 1
7
6 5 4 3 2 1 0
0
```

Sample Output

```
2
1
3
```

PROBLEM F

THE SWEEPER IN CLEANLAND

Cleanland is a city well-known for the civility of its inhabitants. There is no crime, the streets are clean, drivers respect speed limits. In spite of its cleanliness, every day, during the early hours of the morning, a machine is used to clean the curb sides of each street. The city is not too large, so every street is a two-way street, and there is only one machine available to do the job of sweeping the streets. Both sides of each street must be swept. The operator of the sweeping machine is a inhabitant of the city, therefore the curb is traversed in the direction the traffic flows, so each street must be traversed in both directions, cleaning one side at a time.

Your job is to provide the operator with an optimal route, so that no street is traversed more than twice, once in each direction. Of course, one may drive from any point in the city to any other point in that same town. For simplicity, we assume that every street is one block in length, connecting two distinct *corners*. The figure below gives a small example one such town, where is an optimal route.

Input

The input contains several test cases, each case consists of two or more lines. The first line contains two integers, n and m, separated by one space. Integer n, (2<=n<=100), is the number of corners, which are identified by the integers 0, 1, . . . , n−1. Integer m, (0 < m <= n(n−1)/2), is the number of streets. The next m lines describe the m streets: every line contains two integers v and w, separated by one space, corresponding to the corners connected by the street. Note that no two corners are connected by two or more streets. Note also that the ends of each street are distinct. The end of the input is indicated by a line containing precisely two zeros, separated by one space.

Output

For each test case, your program should output one line. The line contains case c: followed by one space, where c is the case number, without leading zeros (the first case is case number 1). Then, separated by precisely one space, an optimal route, described by its corners, in the order they appear in the route. You should not repeat the initial corner as the last corner. All these integers should be printed in the same line, separated by exactly one space.

Sample Input

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

Sample Output

```
case 1: 2 3 4 2 4 3 2 1 0 5 1 5 0 1
case 2: 0 1
```

Notice:

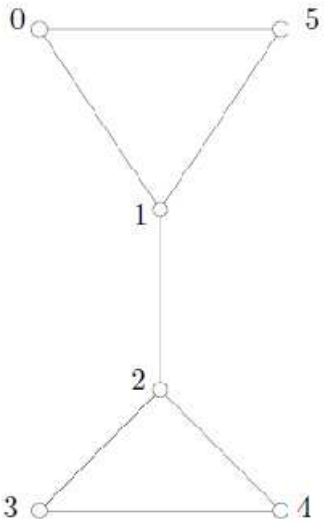


Figure 1: An optimal route: 2 3 4 2 4 3 2 1 0 5 1 5 0 1

PROBLEM G

WORMS

It's the most expected day in the Astounding Confederation of Manzania. Finally, the long awaited games will begin, and the people of this proud nation want to present the best aperture ever made for this kind of events.

Thousands of proudly manzanians will participate in the ceremony that will be held in the historical Manzan´a stadium. Dancers, singers, musicians, lots of color and (real) fireworks all over the place, and of course the national animals can't be absent: the worms, loyal fellows who fought bravely and helped to forge the country. During the show, several performers will march in straight lines inside a rectangular area of the main field, covered with worm costumes of bright colors and several meters long (as the real worms are too small to be visible from the gallery), in a section called "The Dance of the Worms".

No detail has been left unattended. At least that is what you thought before hearing a hard discussion between the president of the organization committee, and the artistic director. The Dance of the Worms is in serious trouble! At the beginning of the dance, all the performers in their worm costumes will start to walk in the already designed roads on the field. But the lines cross, and nobody had payed attention to properly synchronize the start, and some of the worms can collide, bringing chaos and despair to the otherwise perfect ceremony. Now, the artistic director has caught you to help him to solve the mess. Once the worms are inside the field, they can't be stopped (they can't be fired, because of all the legal stuff). They have trained to do the march at 1m/s, with their heavy costume. In order to allow all the worms to walk smoothly on the field, some of them will have to be delayed and will begin the march some seconds later than the others. The delay for each worm has to be an integer number, otherwise they can't count it. The organization, of course, does not want to extend The Dance of the Worms too much ; after all, there are other performances. So, the first thing to know is how much time will the worms be delayed to let all of them march without colliding each other.

You will be given the dimensions of the field where the march will take place, the length of all the worms, and the planned trajectories.

Input

The input contains several test cases. The first line of a test case contains one N indicating the number of worms (2 <= N <= 10). The next line will contain three integers W, H and L indicating respectively the width and length of the field in meters (3 <= W,H <= 50), and the length of the worm costume also in meters (1 <= L <= 10).

Each of the N following lines contains two integers X0, Y0, indicating the coordinates of the origin (X0, Y0) of the straight trajectory for each worm. All trajectories are parallel to one of the borders. The origin will always be located in one border of the field, but never in the corners, and will be different for each worm.

For considering the collisions, you can assume that if one worms arrives at one point at the same time that another worm is leaving that point, no collision will happen. The end of input is indicated by a line containing only one zero.

Output

For each test case in the input, your program must print a single line, containing the sum of all the delays that must be added to the worms to ensure a perfect ceremony.

Sample Input

```
3
8 8 2
2 0
3 0
8 7
3
8 8 1
2 0
3 0
8 7
0
```

Sample Output

```
1
0
```