10930 - A-Sequence

Time limit: 3.000 seconds

For this problem an *A*-sequence is a sequence of positive integers $\mathbf{a_i}$ satisfying $\mathbf{1 \leq a_1 < a_2 < a_3 < ...}$ and every $\mathbf{a_k}$ of the sequence is not the sum of two or more distinct earlier terms of the sequence.

You should write a program to determine if a given sequence it is or it is not an *A*-sequence.

Input

The input consists of a set of lines, each line starts with an integer $\mathbf{2 \leq D \leq 30}$ that indicates the number of integers that the current sequence has. Following this number there is the sequence itself. The sequence is composed by integers, each integer is greater than or equal to **1** and less than or equal to **1000**. The input is terminated by enf of file (EOF).

Output

For each test case in the input you should print two lines: the first line should indicate the number of the test case and the test case itself; in the the second line you should print **This is an A-sequence.**, if the corresponding test case is an A-sequence or **This is not an A-sequence.**, if the corresponding test case is not an A-sequence.

Sample Input

```
2 1 2
3 1 2 3
10 1 3 16 19 25 70 100 243 245 306
```

Sample Output

```
Case #1: 1 2
This is an A-sequence.
Case #2: 1 2 3
This is not an A-sequence.
Case #3: 1 3 16 19 25 70 100 243 245 306
This is not an A-sequence.
```

10883 - Supermean

Time limit: 3.000 seconds

"I have not failed. I've just found 10,000 ways that won't work."

Thomas Edison

Do you know how to compute the mean (or average) of **n** numbers? Well, that's not good enough for me. I want the supermean! "What's a supermean," you ask? I'll tell you. List the **n** given numbers in non-decreasing order. Now compute the average of each pair of adjacent numbers. This will give you **n** - 1 numbers listed in non-decreasing order. Repeat this process on the new list of numbers until you are left with just one number - the supermean. I tried writing a program to do this, but it's too slow. :-(Can you help me?

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each one starts with a line containing **n** (0<**n**<=50000). The next line will contain the **n** input numbers, each one between -1000 and 1000, in non-decreasing order.

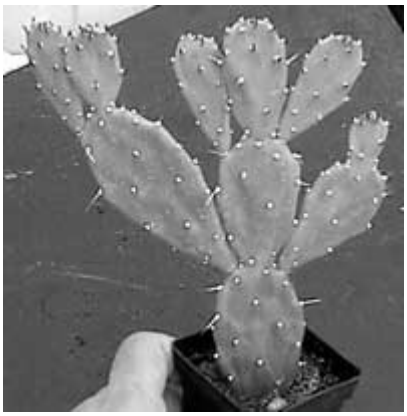
Output

For each test case, output one line containing "Case #**x**:" followed by the supermean, rounded to 3 fractional digits.

Sample Input	Sample Output
4 1 10.4 2 1.0 2.2 3 1 2 3 5 1 2 3 4 5	Case #1: 10.400 Case #2: 1.600 Case #3: 2.000 Case #4: 3.000

10510 - Cactus

Time limit: 3.000 seconds

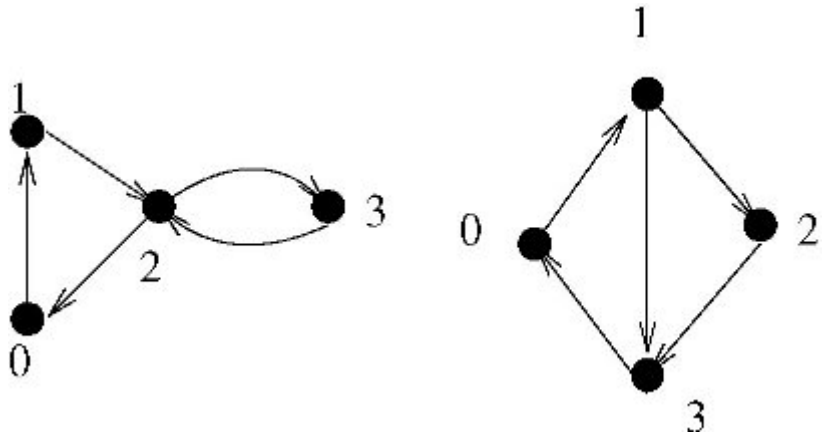


A directed graph is a set **V** of *vertices* and a set of **E** ∈ {**V** x **V**} *edges*. An edge **(u,v)** is said to be directed from **u** to **v** (the edge **(v,u)** has the opposite direction). A directed cycle in a directed graph is a sequence of edges

$(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$

such that $u_{i+1} = v_i$ for $i = 1, \ldots, k-1$, and $u_1 = v_k$. The directed cycle is *simple* if $u_i \neq u_j$ whenever $i \neq j$ (i.e., if it does not pass through a vertex twice).

In a *strongly connected* directed graph, there is for every pair **u,v** of vertices some directed cycle (not necessarily simple) that visits both **u** and **v**.



A directed graph is a *cactus* if and only if it is strongly connected and each edge is part of exactly one directed simple cycle. The first graph is a cactus, but the second one is not since for instance the edge **(0,1)** is in two simple cycles.

The reason for the name is that a "cactus" consists of several simple cycles connected to each other in a tree-like fashion, making it look somewhat like a cactus.

Problem

Write a program that given a directed graph determines if it is a cactus or not. The graph can have several thousand vertices.

Input

The first line contains an integer which is the number of test cases (less than 20). Each test case starts a line with an integer $n \geq 0$ followed by line with an integer $m \geq 0$ giving the number of vertices (**n**) and edges (**m**) in a graph (at most 10,000 of each). The vertices are numbered **0** through **n-1**. The following **m** lines describe the edges as pairs of numbers **u v** denoting an edge directed from **u** to **v**. There will never be more than one edge from **u** to **v** for any pair of vertices **u** and **v**. There are no loops, i.e., no edges from a vertex to itself.

Output

For each test case output a single line with a single string. Output "YES" if the graph is a cactus, and output "NO" if it is not.

Sample Input

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

Sample Output

```
YES
NO
```

11090 - Going in Cycle!!

Time limit: 3.000 seconds

You are given a weighted directed graph with **n** vertices and **m** edges. Each cycle in the graph has a weight, which equals to sum of its edges. There are so many cycles in the graph with different weights. In this problem we want to find a cycle with the minimum mean.

Input

The first line of input gives the number of cases, **N**. **N** test cases follow. Each one starts with two numbers **n** and **m**. **m** lines follow, each has three positive number **a, b, c** which means there is an edge from vertex **a** to **b** with weight of **c**.

Output

For each test case output one line containing "**Case #x:** " followed by a number that is the lowest mean cycle in graph with 2 digits after decimal place, if there is a cycle. Otherwise print "**No cycle found.**".

Constraints

- **$n \leq 50$**
- **$a, b \leq n$**
- **$c \leq 10000000$**

Sample Input

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

Output for Sample Input

```
Case #1: No cycle found.
Case #2: 2.50
```

11071 - Permutation Representation

Time limit: 3.000 seconds

A permutation is a bijection from a set X onto itself. If X is finite, the elements of X are often numbered *1,2,3,...n*. A permutation of a set with five elements is often denoted by

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 5 & 1 & 4 \end{pmatrix}$$

meaning the element 1 is mapped to the element 3 of the set, the element 2 is mapped to the element 2 and so on and so forth. Another way of denoting permutations is to use cycle notation. Cycle notation is not necessarily unique. The following cycle

$$(247)$$

means that the element 2 is mapped to the element 4, the element 4 is mapped to the element 7 and the element 7 is mapped to the element 2. The cycle above could also be written

$$(724)$$

The product of several cycles is evaluated from **right** to **left**. The above permutation can be written as

$$(53)(51)(54)$$

$$(1354)(1)$$

$$(1)(1354)$$

A permutation can be written uniquely as the product of cylces

$$\begin{pmatrix} 1 & 2 & \dots & n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = (1)^{a_1}(12)^{a_2}(123)^{a_3}(1234)^{a_4}\dots(1\dots n)^{a_n}$$

if $0 \le a_i \le i-1$ holds for each exponent a_i. The example permutation can be uniquely written as

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 5 & 1 & 4 \end{pmatrix} = (1)^0(12)^1(123)^2(1234)^2(12345)^2$$

Your task is to compute the a_i's of a given permutation.

Input
The input consists of several test cases. Each test case consists of three lines. The first line contains the number *n, 1 ≤ n ≤ 200000*. The second line contains the elements from 1 to n. The third line contains a mapping for every element from the second line.

Output
For each test case there should be one line of output. Print all the a_i's on a single line separated by one space in the order $a_1...a_n$

Sample Input

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

Sample Output

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

11122 - Tri Tri

Time limit: 3.000 seconds

Given the vertices of two triangles, check whether both of them have any common interior point. No points on the edges or vertices are considered interior to a triangle.

Input

Input starts with an integer *t* (t<= 1000) denoting the number of test cases to follow. Each test case contains 12 integers which are the vertices of the triangles as (*x*, *y*) pair. First three pairs are for one triangle and rest of them are for the other one. None of the triangles will be invalid. There is a blank line before each test case.

Output

For each input, print one line of output. Each line will contain "yes" if there are common interior points between the two triangles, "no" otherwise. See the sample output for exact formatting.

Sample Input

```
2


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


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

Output for Sample Input

```
pair 1: no

pair 2: no
```

338 - Long Multiplication

Time limit: 3.000 seconds

In traditional ``long multiplication'' we determine the product of two integers, *x* and *y*, by multiplying *x* and the individual digits of *y*, in turn, starting with the units digit. The results of these multiplications are arranged appropriately and added, yielding the completed product.

The representation of these operations is usually done in a particular manner. Consider the multiplication of 123 by 95:

```
  123
   95
  ---
  615
1107
-----
11685
```

The numbers to be multiplied, *x* and *y*, are each displayed on a separate line, followed by a horizontal line. The results of multiplying each digit of *y* by *x* are then displayed on separate lines, followed by another horizontal line, and then the final product. In this problem you are to perform a sequence of such multiplications, displaying the results in this traditional representation.

Input

Each line of the input data, except the last, will contain two integers, *x* and *y*, separated by whitespace (one or more blanks and tab characters). Whitespace may also precede the first integer and follow the second integer. Each integer will have no more than 10 digits. The last line of the input data contain only a zero, and marks the end of the input.

Output

For each pair of integers (that is, each input line except the last), perform the multiplication of *x* by *y*, displaying the results in the form shown above and in the examples shown below. Follow the output for each multiplication by a blank line. If there are less than 2 lines of numbers between the horizontal lines, omit them (since in that case they would be superfluous) as well as the second horizontal line, and just output the sum. Display 0 digits only when they are significant.

The number of hyphens in the first horizontal line should be the same as the number of digits in the larger of *x* and *y*. The number of hyphens in the second horizontal line, if it is produced, should be the same as the number of digits in the product of *x* and *y*.

Sample Input

```
   4 7
135  46
    12345   862
0 123456789
0
```

Sample Output

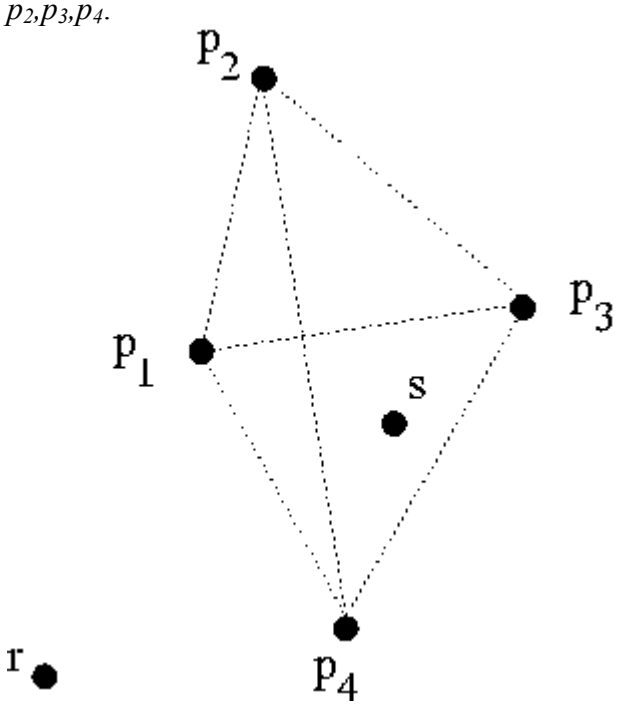
```
 4
 7
 -
28

 135
  46
 ---
 810
540
----
6210

   12345
     862
   -----
   24690
  74070
 98760
--------
10641390

        0
123456789
---------
        0
```

11072 - Points

Time limit: 3.000 seconds

In this problem you will be given a set of points in the Euclidian plane. The number of points in the set will never exceed *100000*. The coordinates of these points will be integer coordinates and will have an absolute value smaller than *10000*. There will be no identical points in the first set. Then you will be given a second set of points. For each point in the second set you will have to determine whether it lies in a triangle spanned by three points in the first set. A point lying on the edge of a triangle is considered to be "inside" the triangle.

In the following example the points p_1,p_2,p_3,p_4 belong to the first set. The points *r* and *s* belong to the second set. The point *r* isn't contained in any triangle spanned by three points of the first set. The point *s* is contained in two triangles. For example, the triangle spanned by p_2,p_3,p_4.



Input

You will be given several testcases. A testcases consists of the number of points *p*, $3 \leq p \leq 100000$ in the first set. It is followed by *p* pairs of numbers, each describing a point of the first set, the first number of a pair denoting the *x*-coordinate of the point, the second the *y*-coordinate. Each pair is on a seperate line. There may be colinear points in the first set. The next number in the input gives you the number of points *r* in the second set. It is followed by *r* pairs of numbers, each describing a point, each on a separate line. The first number of a pair being the *x*-coordinate, the second number being the *y*-coordinate of the point. All coordinates in the input will be integer coordinates.

Output

For each point in the second set, output if the point lies in a triangle spanned by three points of the first set. If the point lies inside a triangle output **inside** otherwise output **outside**.

Sample input

```
4
0 0
4 4
0 4
4 0
6
2 2
4 4
1 1
0 2
0 10
10 0
```

Sample output

```
inside
inside
inside
inside
outside
outside
```

11096 - Nails

Time limit: 3.000 seconds

Arash is tired of hard working, so he wants to surround some nails on the wall of his room by a rubber ribbon to make fun of it! Now, he wants to know what will be the final length of the rubber ribbon after surrounding the nails. You must assume that the radius of nails and rubber ribbon is negligible.

Input

The first line of input gives the number of cases, **N**. **N** test cases will follow. Each test case starts with a line containing two integers, the initial length of rubber ribbon and the number of nails **0<n<=100,** respectively. Each of next **n** lines contains two integers denoting the location of a nail. There will be a blank line after each test-case.

Output

Your program must output the final length of rubber ribbon precise to 5 decimal digits.

Sample Input	Output for Sample Input
2 2 4 0 0 0 1 1 0 1 1 5 4 0 0 0 1 1 0 1 1	4.00000 5.00000

653 - Gizilch

Time limit: 3.000 seconds

The game of gizilch has very simple rules. First 100 grapes are labeled, in nontoxic ink, with the numbers 1 to 100. Then, with a cry of ``GIZILCH!", the referee fires the grapes up into the air with a giant gizilcher. The two players, who each start with a score of ``1", race to eat the falling (or, shortly thereafter, fallen) grapes and, at the same time, multiply their scores by the numbers written on the grapes they eat. After a minute, the hungry squirrels are let loose to finish the remaining grapes, and each contestant reports his score, the product of the numbers on the grapes he's eaten. The unofficial winner is the player who announces the highest score.

Inevitably, though, disputes arise, and so the official winner is not determined until the disputes are resolved. The player who claims the lower score is entitled to challenge his opponent's score. The player with the lower score is presumed to have told the truth, because if he were to lie about his score, he would surely come up with a bigger better lie. The challenge is upheld if the player with the higher score has a score that cannot be achieved with grapes not eaten by the challenging player. So, if the challenge is successful, the player claiming the lower score wins.

So, for example, if one player claims 343 points and the other claims 49, then clearly the first player is lying; the only way to score 343 is by eating grapes labeled 7 and 49, and the only way to score 49 is by eating a grape labeled 49. Since each of two scores requires eating the grape labeled 49, the one claiming 343 points is presumed to be lying.

On the other hand, if one player claims 162 points and the other claims 81, it is possible for both to be telling the truth (e.g. one eats grapes 2, 3 and 27, while the other eats grape 81), so the challenge would not be upheld.

Unfortunately, anyone who is willing to referee a game of gizilch is likely to have himself consumed so many grapes (in a liquid form) that he or she could not reasonably be expected to perform the intricate calculations that refereeing requires. Hence the need for you, sober programmer, to provide a software solution.

Input

Pairs of unequal, positive numbers, with each pair on a single line, that are claimed scores from a game of gizilch.

Output

Numbers, one to a line, that are the winning scores, assuming that the player with the lower score always challenges the outcome.

Sample Input

```
343 49
3599 610
62 36
```

Sample Output

```
49
610
62
```

Clarification:

The rules for deciding the winner of a game of gizilch are, first, if both players might be telling the truth, the larger score wins. Second, if the player with the lower score cannot be telling the truth, the player with the higher score wins. Finally, if neither of the previous two conditions holds, the lower score wins.

673 - Parentheses Balance

Time limit: 3.000 seconds

You are given a string consisting of parentheses `()` and `[]`. A string of this type is said to be *correct*:

(a)
if it is the empty string

(b)
if A and B are correct, AB is correct,

(c)
if A is correct, `(`A`)` and `[`A`]` is correct.

Write a program that takes a sequence of strings of this type and check their correctness. Your program can assume that the maximum string length is 128.

Input

The file contains a positive integer *n* and a sequence of *n* strings of parentheses `()` and `[]`, one string a line.

Output

A sequence of `Yes` or `No` on the output file.

Sample Input

```
3
([])
((([()])))
([()[]()])()
```

Sample Output

```
Yes
No
Yes
```

190 - Circle Through Three Points

Time limit: 3.000 seconds

Your team is to write a program that, given the Cartesian coordinates of three points on a plane, will find the equation of the circle through them all. The three points will not be on a straight line.

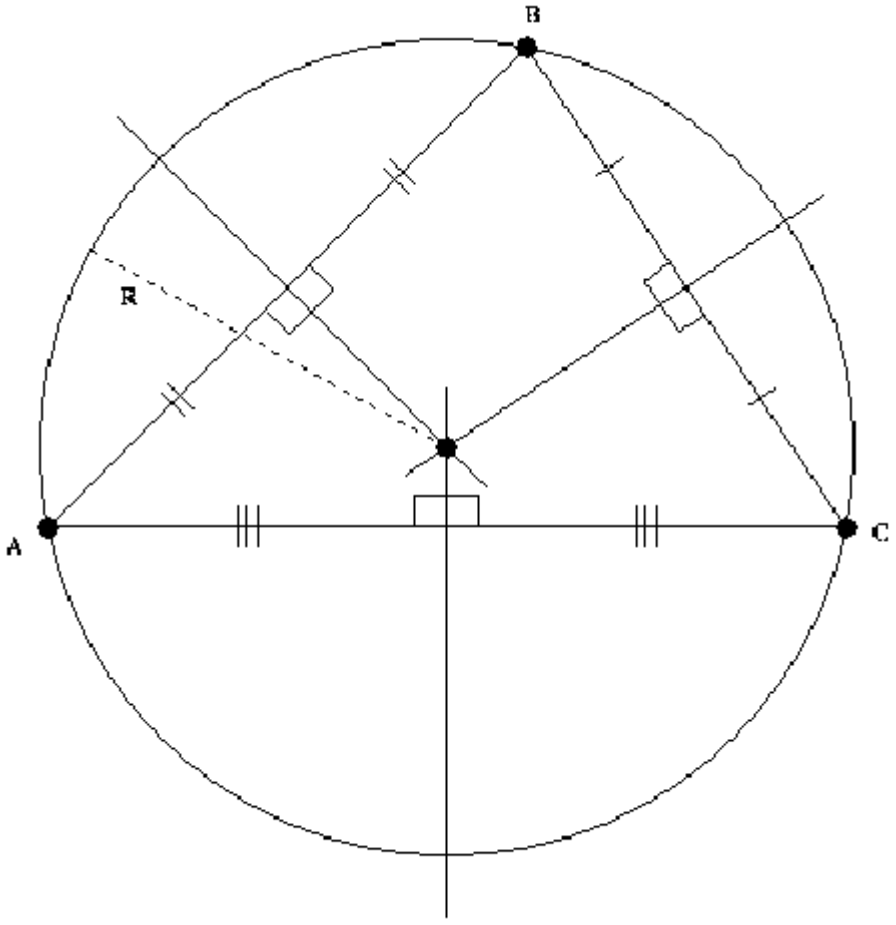
The solution is to be printed as an equation of the form

$$(x-h)^2+(y-k)^2=r^2 \qquad (1)$$

and an equation of the form

$$x^2+y^2+cx+dy+e=0 \qquad (2)$$

Each line of input to your program will contain the *x* and *y* coordinates of three points, in the order A_x, A_y, B_x, B_y, C_x, C_y. These coordinates will be real numbers separated from each other by one or more spaces.



Your program must print the required equations on two lines using the format given in the sample below. Your computed values for *h*, *k*, *r*, *c*, *d*, and *e* in Equations 1 and 2 above are to be printed with three digits after the decimal point. Plus and minus signs in the equations should be changed as needed to avoid multiple signs before a number. Plus, minus, and equal signs must be separated from the adjacent characters by a single space on each side. No other spaces are to appear in the equations. Print a single blank line after each equation pair.

Sample input

```
7.0 -5.0 -1.0 1.0 0.0 -6.0
1.0 7.0 8.0 6.0 7.0 -2.0
```

Sample output

```
(x - 3.000)^2 + (y + 2.000)^2 = 5.000^2
x^2 + y^2 - 6.000x + 4.000y - 12.000 = 0

(x - 3.921)^2 + (y - 2.447)^2 = 5.409^2
x^2 + y^2 - 7.842x - 4.895y - 7.895 = 0
```