Problem A
Problem : Budget Movie

Input File: movie.in
Output File: movie.out

As director of the upcoming blockbuster movie *Deep Code 9*, your employer *21st Century Fox* has assigned you the task of selecting the actors and actresses.

The problem is that in the movie about 10 male and 10 female characters occur, and - with the tiny budget that you have been given - you simply cannot afford that many actors and actresses!
Looking closely at the movie script, however, you wonder if some of the characters could be played by the same person. For example, if Mr. Programmer and Mr. Hero never appear together at the same time during the movie, one actor can play both roles!

You tell your boss your idea, and he agrees, as long as you adhere to the following rules:

1. Male characters can only be played by male actors, and female characters can only be played by female actresses.
2. Each character must be played by one actor/ actress throughout the whole movie. Changing the person who plays a given character during the movie is not allowed.
3. When two characters ever appear together at the same time during the movie, they must be played by different actors/ actresses.

Given these restrictions, your job is to determine the minimum number of actors and actresses needed to produce the movie.

Input Specification

The input consists of several movie descriptions. Each description starts with one line that contains three integers M, F, S that specify the number of male (1<=M<=10) and female (1<=F<=10) characters occurring in the movie and the number of scenes that the movie has (1<=S<=100).
On the second and third line, the names of the male and female characters are given.
Then S lines follow, each line describing one scene. Each of these lines contains the number of people who occur in the scene and the list of their names.

Output Specification

For each movie description, output three lines:

- a line saying: "Movie #*n*" where *n* is the number of the movie
- a line saying: "You need *x* actors and *y* actresses." where *x* and *y* is the solution. Use singular (actor, actress), if $x = 1$ or $y = 1$.
- a blank line

See the sample output below for the correct output format.

Sample Input

```
1 1 1
Donald
Daisy
2 Donald Daisy
```

4 3 6
Tarzan Jim John Tom
Lucy Cynthia Jane
3 Jim John Tom
2 Tarzan Lucy
2 Jane Cynthia
2 Jim Jane
2 Tarzan Jim
2 Tarzan Jane
0 0 0

Sample Output

Movie #1
You need 1 actor and 1 actress.

Movie #2
You need 3 actors and 2 actresses.

Problem B
Let's compute

Input file: compute.in
Output file: compute.out

You are to write a program that evaluates assignment statements. Assume that there is one (and only one) statement on each line of the input, and each assignment statement is terminated by a semicolon. The following are some of possible assignment statements:

```
      length = 4 +6;
       width=3+12*10;
     area =length    *width - b;
    total   =100 - y- (width+111*222)    /area + x;
       area= area +5 -  3 *4^ (    width  - 3*(length+30
))+100;
          x  =     area +(total-area/  width ) ;
```

Here are some assumptions and rules:

(1) All variables are 1-10 characters long, and only lowercase letters are used.

(2) All values are integers, and all numbers are in the range of -32768 to 32767. (Such numbers are stored in 16 bits of memory.)

(3) / is the integer division operator. So 7/2 = 3 (truncated to integer), and 100/7 = 14.

(4) ^ is the exponential operator. For example, 2^5 = 2*2*2*2*2 = 32.

(5) An expression may contain both integer literals and variables.

(6) Any number of spaces may be used to separate different items in a statement.

(7) All statements are 1 - 80 characters long.

(8) All variables have a default value of 0.

(9) All statements are valid.

(10) First line of the input contains a positive integer indicating the number of statements to be evaluated.

Your program should read and evaluate all statements and print a table that shows all variables, in alphabetical order and aligned

to the left, that appear in the input data along with their corresponding final values aligned to right.

Sample Input

```
6
 length = 4 +6;
  width=3+12*10;
area =length    *width - b;
total   =100 - y- (width+111*222)    /area + x;
 area= area +5 -  3 *4^ (    width  - 3*(length+30    ))+100;
     x  =     area +(total-area/  width ) ;
```

Sample Output

```
Variable            Value
=========================

area                 1143
b                       0
length                 10
total                  80
width                 123
x                     1214
y                       0
```

Problem C
Problem : Subsequence

Input file: subseq.in
Output file: subseq.out

This problem deals with finding the longest common subsequence between a given set of sequences. A subsequence of a given sequence is simply the given sequence with some elements, possibly none, removed. For example, (2,4,1,5,7) is a subsequence of (9,2,3,4,1,4,5,12,7,10,9). Every sequence is a subsequence of itself; the empty sequence is a subsequence of every sequence. Given two sequences A and B, we say that X is a common subsequence of A and B, if C is subsequence of both A and B. Given three sequences A, B, and C, we say that X is a common subsequence of A, B and C, if X is a subsequence of A, is a subsequence of B, and is a subsequence of C. The longest common subsequence of a given trio of sequences is the maximum-length common subsequence of the three sequences.

Input

The input file will consist of several problem instances. The first line of the file will include a single integer, indicating the number of problem instances that will follow. Each problem instance consists of three lines, each line consisting of a sequence of positive integers, separated by space and terminated with a zero (the final zero is not considered part of the sequence). The length of the sequence will not be greater than 50.

Output

For each problem instance, your program should output a single line consisting of a single integer, representing the length of the longest common subsequence of the two given sequences.

Sample Input

```
6
2 1 3 5 3 8 5 9 4 0
1 2 3 4 8 9 12 14 0
4 1 2 3 4 8 12 9 14 9 2 0
3 1 2 4 5 6 0
3 1 2 4 5 6 0
3 1 2 4 5 6 0
3 2 1 9 10 3 8 4 9 2 3 4 0
1 3 2 9 4 2 4 2 3 4 5 2 4 0
1 3 2 3 9 3 2 2 4 2 2 9 4 2 3 4 4 2 1 0
2 1 4 2 3 9 2 3 4 2 4 0
2 2 2 2 4 0
1 2 3 4 2 4 4 9 2 4 2 3 4 0
2 1 3 2 4 9 2 3 0
8 9 8 9 8 0
5 7 6 7 9 7 5 0
1 2 3 4 5 0
6 7 8 9 10 0
3 0
```

Sample Output

```
4
6
7
5
1
0
```

Problem D
Problem : easy

Input file: easy.in
Output file: easy.out

You are given a file containing several data sets. The first line of the file contains a single integer N, and the rest of the file contains one integer per line. The number N refers to the number of data sets in the file. Each data set consists of a number of lines, each line containing a single integer, terminated by a line containing the integer 0. For each data set, we would like to know whether the data set contains a data element *x* with the property that *2x* is equal to the sum of the other data elements (not including *x*). If the data set includes more than one element with this property then the data element which occurs first is the one that should be printed. If no such element exists, then the number 0 should be printed.

Input

The input file will begin with an input line consisting of a single integer N. The number N refers to the number of data sets in the file. Each data set consists of a number of lines, each line consisting of a single integer (in the range -30000…30000), and is terminated by a line containing only the number 0.

Output

For each data set, output a single line containing the integer *x*.

Sample Input

```
4
-5
15
42
-6
32
18
30
0
2
8
13
6
10
0
4
6
-8
19
-9
12
0
0
```

Sample Output

```
42
13
0
0
```

Problem E
Problem : Anagrams by Stack

Input file: stack.in
Output file: stack.out

A school teacher wants to explain the concept of a stack and its operations to his classroom. He wants a program to show them how anagrams can result from sequences of stack operations: 'push' and 'pop' .He'll be illustrating on 4-letter words. There are two sequences of stack operators which can convert TROT to TORT:

```
[
i i i i o o o o
i o i i o o i o
]
```

where *i* stands for Push and *o* stands for Pop. Given pairs of words, your program should produce sequences of stack operatons which convert the first word to the second.

Input

The input file will consist of several pairs of input lines. The first line of the file will include a single integer, indicating the number of pairs of input lines that will follow. The first line of each pair of input lines is to be considered as a 4-letter source word .The second line is the 4-letter target word.

Output

For each input pair, your program should produce a sorted list of valid sequences of *i* and *o* which produce the target word from the source word. Each list should be delimited by

```
[
]
```

and the sequences should be printed in "dictionary order". Within each sequence, each *i* and *o* is followed by a single space and each sequence is terminated by a new line.

Process

A stack is a data storage and retrieval structure permitting two operations:

Push - to insert an item and
Pop - to retrieve the most recently pushed item

We will use the symbol *i* (in) for push and *o* (out) for pop operations for an initially empty stack of characters. Given an input word, some sequences of push and pop operations are valid in that every character of the word is both pushed and popped, and futhermore, no attempt is ever made to pop the empty stack. For example, if the word FOO is input, then the sequence:

`i i o i o o` is valid, but
`i i o` is not (it's too short), neither is
`i i o o o i` (there's an illegal pop of an empty stack)

Valid sequences yield rearrangements of the letters in an input word. For example, the input word FOO and the sequence `i i o i o o` produce the anagram OOF. So also would the sequence `i i i o o o`.

You are to write a program to input pairs of words and output all the valid sequences of `i` and `o` which will produce the second member of each pair from the first.

Sample Input

```
3
mada
adam
long
nice
eric
rice
```

Sample Output

```
[
i i i i o o o o
i i o i o i o o
]
```

[
]
[
i i o i o i o o
]

Problem F
Problem : Control Break

Input file: break.in
Output file: break.out

BIGIANT UNIVERSITY reports payroll expenses to the administration giving totals by department within school. For instance, within the school of arts and sciences totals would be accumulated for the Psychology, Economics, Philosophy, and other departments.

Your task is to write a program, which will accumulate total payroll expenses by department within school and report them. You will not list individual pay amounts, only the totals by department and by school.

You must report the total number of employees and total pay amount by department and by school, and give a grand total for the entire university. The data from which these totals are to be calculated are arranged in order by department within school. This process is known as "control break processing" because a control break is said to have occurred each time a department or school changes.

The following is an example of the type of data, which you will be processing: (End of input data is indicated by a line containing all nines.) The columns for the data are specified below:

SCHOOL NUMBER	**DEPARTMENT NUMBER**	**INDIVIDUAL PAY AMOUNT**
Column 1	Cols. 2 & 3	Cols. 4-10

Sample Data:

```
1010050000
1010025050
1010037825
1020078550
1020046520
2010075000
2010049675
2020120000
2020037800
2030074250
2040037500
2040065050
3010091613
3010085544
3020057515
3020110525
9999999999
```

For the above sample data, the printed report would appear as follows:

BIGIANT UNIVERSITY
PAYROLL EXPENSE REPORT

SCHOOL	DEPARTMENT	NO. OF EMPLOYEES	PAYROLL
1	01	3	1,128.75
1	02	2	1,250.70
	SCHOOL TOTALS	5	2,379.45
2	01	2	1,246.75
2	02	2	1,578.00
2	03	1	742.50
2	04	2	1,025.50
	SCHOOL TOTALS	7	4,592.75
3	01	2	1,771.57
3	02	2	1,680.40
	SCHOOL TOTALS	4	3,451.97
	**UNIVERSITY TOTALS*	16	10,424.17

All titles and identifying information should be identical to the given output. Output should have SCHOOL, DEPARTMENT, and NO. OF EMPLOYEES columns right aligned. The PAYROLL column should be decimally aligned. The SCHOOL and UNIVERSITY TOTAL lines should also contain entries that are right aligned and decimally aligned with the data above them. All PAYROLL numbers, should include commas where appropriate. All titles (including the required report title) and identifying information should be in upper case letters. Blank lines should be included after the report title, the heading line, and each school total line. Horizontal spacing between columns may be approximate; however, data should be under appropriate headings.

Problem G
Problem : Intersecting Line Segments

Input file: lines.in
Output file: lines.out

The purpose of this problem is to determine whether two given line segments intersect. Two line segments are considered to intersect if they have at least one endpoint in common. A line segment is defined by its two endpoints: (x1,y1) and (x2,y2). The input file will consist of a number of pairs of linear segments. For each pair, if the two line segments intersect, then you will print "yes", if they do not, you will print "no."

Input

The input file will consist of several problem instances, each problem instance representing a pair of line segments. The first line of the file will include a single integer, indicating the number of problem instances that will follow. Each problem instance consists of two lines: the first line contains four positive integer numbers (in the range 0…1000): x1, y1, x2, y2, separated by space, representing the two endpoints of the first line segment; the second line contains four integer numbers (in the range 0…100): x1, y1, x2, y2, separated by space, representing the two endpoints of the second line segment.

Output

For each problem instance, your program should output a single line consisting of the word "yes" if the two line segments intersect and the word "no" if the two line segments do not intersect.

Sample Input

```
2
100 100 200 200
200 100 100 200
50 60 80 90
40 50 45 70
```

Sample Output

```
yes
no
```

Problem H
Problem : Metro

Input file: metro.in
Output file: metro.out

The Cairo Metro Authority has asked you to write a program to compute the average time necessary to travel between any two metro stations in Cairo. You are given the number of stations; for every pair of stations that are directly adjacent, you are given the average time needed to travel from one station to the other. (Mathematically, this is a weighted undirected graph showing which stations are connected to which). Connections are assumed to be symmetric, i.e. the length of the connection from a to b will always be the same as the length from b to a. For every station in Cairo, you are to find the shortest path from this station to all other stations. You are then to find the average shortest path over the entire city.

There might be some pairs of stations that have no path between them; such pairs of stations should be excluded from the average computation.

Example

Suppose we have 7 stations {a,b,c,d,e,f,g} whose connections are shown in the following table:

	a	b	c	d	e	f	g
A	--	2	5	--	--	--	--
B	2	--	3	4	--	--	--
C	5	3	--	1	2	--	--
D	--	4	1	--	--	--	--
E	--	--	2	--	--	--	--
F	--	--	--	--	--	--	3
G	--	--	--	--	--	3	--

Solution to Example

The following table shows the length of the shortest path between every pair of stations; note that some pairs of stations have no path between them:

	a	b	c	d	e	f	g
a	--	2	5	6	7	--	--
b	2	--	3	4	5	--	--
c	5	3	--	1	2	--	--
d	6	4	1	--	3	--	--
e	7	5	2	3	--	--	--
f	--	--	--	--	--	--	3
g	--	--	--	--	--	3	--

The average shortest path length, over all pairs of stations which have paths between them, is 3.7272.

Input

The input file will consist of several problem instances. The first line of the file will include a single integer, indicating the number of problem instances that will follow. Each problem instance is structured as follows: the first line is an integer indicating the number of stations; the second line is an integer *n* indicating the number of connections *m*; this is followed by *m* lines, each line representing a connection. Each connection consists of three integers: the first two integers representing the two stations numbers, and the third integer representing the length of the connection. The maximum number of stations in any problem instance will be 40. The maximum connection length in any problem instance will be 50.

Output

For each problem instance, your program should output a single line consisting of a single floating point number, representing the average length of the shortest path over all pairs of stations which have paths. This floating point number should be printed with exactly four decimal places; anything after the fourth decimal place should be truncated.

Sample Input

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

Sample Output

```
3.7272
2.0000
```

Problem I
Problem : Turning Mirrors

Input File: mirrors.in
Output File: mirrors.out

It's 3000 years ago. The Pharao's pyramid has just been finished, and now the interior needs to be worked on. However, it's pitch dark inside, and the problem of providing light for the workers has to be solved.

Electric light hasn't been invented yet, and lighting a fire inside the pyramid isn't a good solution either, because the interior should stay clean of smoke and ash. Therefore, the senior architect's proposal is to use mirrors to redirect sunlight from the entrance to the construction sites. From time to time, after work had been finished at one place, the mirrors will have to be realigned in order to direct the light to another place.

For this task of realigning the mirrors, an algorithm is needed. The input will be a two-dimensional map of one floor. The map displays the origin and the desired destination of the light, all obstacles and all mirrors, and the current path of the light. The output should be another map in which the mirrors are realigned appropriately (if neccessary) so that the light reaches the destination.

As the Pharao's chief software engineer, it's your job to design this algorithm. The "computers" - a group of mathematically trained workers - will execute the program manually later. (Now you know why it took decades to build the pyramids.)

Input Specification

The input file consists of one or more test cases. Each test case starts with a line that contains two integers: the number of rows R and the number of columns C of the map.
Then a map of the current situation follows, given as R rows of C characters each.
The possible characters are 'O', 'D', '/', '\', '-', '|', '+', '#' and '.'. They stand for:

O	origin of light
D	destination of light
/	mirror in position 1
\	mirror in position 2
#	obstacle
.	free space
|	light ray moving vertically
-	light ray moving horizontally
+	crossing light rays

The origin of light 'O' will always be on the border. The remaining border will consist entirely of '#' characters. Light always travels vertically or horizontally, and reflections are always 90-degree turns.

Output Specification

For each test case, output the same map with the following two modifications:

- Turn mirrors where it is neccessary, i.e. replace / by \ or vice versa.
- Redraw the light ray, i.e. change the characters `|,-,+ and .` accordingly, so that it goes from O to D.

The characters O, D, and # always remain unchanged.

Sample Input

```
4 6
###O##
#..|.#
#D.\-#
######
11 49
##O#############################################
##|#.........................................#
#.|............................/..\..........#
#.|.....\.....................................#
#.|..............#####........................#
#.|........./........................./.........#
#.|..............#####........................#
#.\------------------------------\.............#
#......###...###.................|.............#
#......###.D.###-----------------/.............#
#############################################
0 0
```

Sample Output

```
###O##
#..|.#
#D-/.#
######

##O#############################################
##|#.........................................#
#.|............................/--\..........#
#.|.....\.....................|..|..........#
#.|..............#####.......|..|..........#
#.|........./---------------------+--/.........#
#.|........|.....#####.........|.............#
#.\--------+--------------------/.............#
#......###.|.###.................................#
#......###.D.###................./.............#
#############################################
```