Greetings From Globussoft

- Given below are 5 Programming questions, you have to solve any 3 out of 5 questions.

- These 5 questions you can attempt in any technology like C/C++, java, .Net, PHP

- To solve these 3 questions you've max. 3 hours.

- While Solving these questions you are not allowed to use any **Search Engine** like Google, Yahoo, Bing …

All the best for your test

Globussoft

QUESTION – 1

Let us consider a triangle of numbers in which a number appears in the first line, two numbers appear in the second line etc. Develop a program which will compute the largest of the sums of numbers that appear on the paths starting from the top towards the base, so that:

- on each path the next number is located on the row below, more precisely either directly below or below and one place to the right;
- the number of rows is strictly positive, but less than 100;
- all numbers are positive integers between O and 99.

Take care about your fingers, do not use more than **256** bytes of code.

Input

In the first line integer n - the number of test cases (equal to about 1000). Then n test cases follow. Each test case starts with the number of lines which is followed by their content.

Output

For each test case write the determined value in a separate line.

Example

Input:

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

Output:

```
5
9
```

QUESTION – 2

Preparing a problem for a programming contest takes a lot of time. Not only do you have to write the problem description and write a solution, but you also have to create difficult input files. In this problem, you get the chance to help the problem setter to create some input for a certain problem.
For this purpose, let us select the problem which was not solved during last year's local contest. The problem was about finding the optimal binary search tree, given the probabilities that certain nodes are accessed. Your job will be: given the desired optimal binary search tree, find some access probabilities for which this binary search tree is the unique optimal binary search tree.

Don't worry if you have not read last year's problem, all required definitions are provided in the following.
Let us define a **binary search tree** inductively as follows:

- The empty tree which has no node at all is a binary search tree
- Each non-empty binary search tree has a root, which is a node labelled with an integer, and two binary search trees as left and right subtree of the root
- A left subtree contains no node with a label ≥ than the label of the root
- A right subtree contains no node with a label ≤ than the label of the root

Given such a binary search tree, the following **search procedure** can be used to locate a node in the tree:
Start with the root node. Compare the label of the current node with the desired label. If it is the same, you have found the right node. Otherwise, if the desired label is smaller, search in the left subtree, otherwise search in the right subtree.
The **access cost** to locate a node is the number of nodes you have to visit until you find the right node. An **optimal binary search tree** is a binary search tree with the minimum expected access cost.

Input Specification

The input file contains several test cases.
Each test case starts with an integer n ($1 \leq n \leq 50$), the number of nodes in the optimal binary search tree. For simplicity, the labels of the nodes will be integers from *1* to n. The following n lines describe the structure of the tree. The i-th line contains the labels of the roots of the left and right subtree of the node with label i (or -1 for an empty subtree). You can assume that the input always defines a valid binary search tree.
The last test case is followed by a zero.

Output Specification

For each test case, write one line containing the access frequency for each node in increasing order of the labels of the nodes. To avoid problems with floating point precision, the frequencies should be written as integers, meaning the access probability for a node will be the frequency divided by the sum of all frequencies. Make sure that you do not write any integer bigger than $2^{63} - 1$ (the maximum value fitting in the C/C++ data type *long long* or the Java data type *long*). Otherwise, you may produce any solution ensuring that there is exactly one optimal binary search tree: the binary search tree given in the input.

Sample Input

```
3
-1 -1
1 3
-1 -1
10
-1 2
-1 3
-1 4
-1 5
-1 6
-1 7
-1 8
-1 9
```

```
-1 10
-1 -1
0
```

Sample Output

```
1 1 1
512 256 128 64 32 16 8 4 2 1
```

QUESTION – 3

In the movie "Blues Brothers", the orphanage where Elwood and Jake were raised may be sold to the Board of Education if they do not pay 5000 dollars in taxes at the Cook County Assessor's Office in Chicago. After playing a gig in the Palace Hotel ballroom to earn these 5000 dollars, they have to find a way to Chicago. However, this is not so easy as it sounds, since they are chased by the Police, a country band and a group of Nazis. Moreover, it is 106 miles to Chicago, it is dark and they are wearing sunglasses.
As they are on a mission from God, you should help them find the safest way to Chicago. In this problem, the safest way is considered to be the route which maximises the probability that they are not caught.

Input Specification

The input file contains several test cases.
Each test case starts with two integers *n* and *m* ($2 \le n \le 100$, $1 \le m \le n*(n-1)/2$). *n* is the number of intersections, *m* is the number of streets to be considered.
The next *m* lines contain the description of the streets. Each street is described by a line containing 3 integers *a*, *b* and *p* ($1 \le a, b \le n$, $a \ne b$, 1 ≤ p &le 100): *a* and *b* are the two end points of the street and *p* is the probability in percent that the Blues Brothers will manage to use this street without being caught. Each street can be used in both directions. You may assume that there is at most one street between two end points.
The last test case is followed by a zero.

Output Specification

For each test case, calculate the probability of the safest path from intersection *1* (the Palace Hotel) to intersection *n* (the Honorable Richard J. Daley Plaza in Chicago). You can assume that there is at least one path between intersection *1* and *n*.
Print the probability as a percentage with exactly 6 digits after the decimal point. The percentage value is considered correct if it differs by at most 10^{-6} from the judge output. Adhere to the format shown below and print one line for each test case.

Sample Input

```
5 7
5 2 100
3 5 80
2 3 70
2 1 50
3 4 90
4 1 85
3 1 70
0
```

Sample Output

```
61.200000 percent
```

QUESTION – 4

In a billiard table with horizontal side **a** inches and vertical side **b** inches, a ball is launched from the middle of the table. After **s** > 0 seconds the ball returns to the point from which it was launched, after having made **m** bounces off the vertical sides and **n** bounces off the horizontal sides of the table. Find the launching angle **A** (measured from the horizontal), which will be between 0 and 90 degrees inclusive, and the initial velocity of the ball.

Assume that the collisions with a side are elastic (no energy loss), and thus the velocity component of the ball parallel to each side remains unchanged. Also, assume the ball has a radius of zero. Remember that, unlike pool tables, billiard tables have no pockets.

Input

Input consists of a sequence of lines, each containing five nonnegative integers separated by whitespace. The five numbers are: **a**, **b**, **s**, **m**, and **n**, respectively. All numbers are positive integers not greater than 10000.

Input is terminated by a line containing five zeroes.

Output

For each input line except the last, output a line containing two real numbers (accurate to two decimal places) separated by a single space. The first number is the measure of the angle **A** in degrees and the second is the velocity of the ball measured in inches per second, according to the description above.

Example

Input:

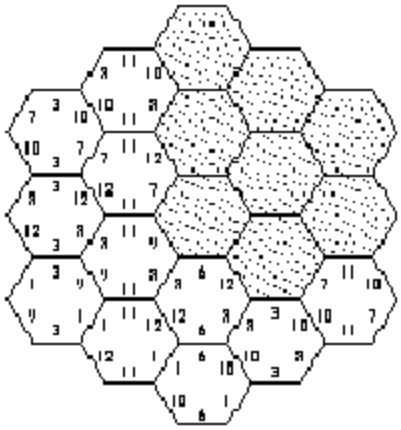
```
100 100 1 1 1
200 100 5 3 4
201 132 48 1900 156
0 0 0 0 0
```

Output:

```
45.00 141.42
33.69 144.22
3.09 7967.81
```

QUESTION – 5

Consider a game board consisting of 19 hexagonal fields, as shown in the figure below. We can easily distinguish three main directions in the shape of the board: from top to bottom, from top-left to bottom-right, and from top-right to bottom-left. For each of these primary directions, the board can be viewed as a series of rows, consisting of 3, 4, 5, 4, and 3 fields, respectively.



The game board has to be completely covered using a set of hexagonal pieces. Each piece carries three numbers, one for every primary board direction. Only three different numbers are used for each direction. Every possible combination of three numbers for all three directions is assigned to a piece, leading to a set of 27 unique pieces. (The board in the above figure is still in the process of being covered.)

The score of a board is calculated as the sum of all 15 row scores (5 rows for each primary direction). The row scores are calculated as follows: if all pieces in a row carry the same number for the direction of the row, the row score is this number multiplied by the number of pieces in the row. Otherwise (the pieces carry different numbers in the row direction) the row score is zero. Note that the pieces may not be rotated. For example, the score of the leftmost row in the figure is $3 \cdot 3 = 9$, the score of the row to its right is $4 \cdot 11 = 44$.

While in the real game the pieces are chosen randomly and the set of pieces is fixed, we are interested in the highest possible score for a given set of numbers for each direction, when all pieces in a row carry the same number for the direction of the row. This means you have to choose those 19 pieces that result in the highest score under the constraint that each row has a score greater than zero.

Input

The first line of the input file contains an integer **n** which indicates the number of test cases. Each test case consists of three lines containing three different positive integers each. Each of these three lines contains the numbers for a single primary direction. From these numbers the set of pieces is generated.

Output

For each test case output a line containing the number of the case (`"Test #1"`, `"Test #2"`, etc.), followed by a line containing the highest possible score for the given numbers. Add a blank line after each test case.

Example

Input:

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

Output:

```
Test #1
308
```