ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 1 – IP Address
Source Program: IPAddress.cpp, IPAddress.java, IPAddress.c, IPAddress.pas
Input file: IPAddress.in **Output file:** IPAddress.out

Suppose you are reading byte streams from any device, representing IP addresses. Your task is to convert a 32 characters long sequence of '1s' and '0s' (bits) to a dotted decimal format. A dotted decimal format for an IP address is form by grouping 8 bits at a time and converting the binary representation to decimal representation. Any 8 bits is a valid part of an IP address. To convert binary numbers to decimal numbers remember that both are positional numerical systems, where the first 8 positions of the binary systems are:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Input
The input file, "IPAddress.in", will have a number N (1<=N<=9) in its first line representing the number of streams to convert. N lines will follow.

Output
The output file "IPAddress.out" must have N lines with a doted decimal IP address. A dotted decimal IP address is formed by grouping 8 bit at the time and converting the binary representation to decimal representation.

Input Example:
```
4
00000000000000000000000000000000
00000011100000001111111111111111
11001011100001001110010110000000
01010000000100000000000000000001
```

Output Example:
```
0.0.0.0
3.128.255.255
203.132.229.128
80.16.0.1
```

ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 2 – Boolean Expressions
Source Program: BooleanE.cpp, BooleanE.java, BooleanE.c, BooleanE.pas
Input file: BooleanExp.in **Output file:** BooleanExp.out

Problem Description: The objective of the program you are going to produce is to evaluate boolean expressions as the one shown next:

Expression: (V | V) & F & (F | V)

where **V** is for True, and **F** is for False. The expressions may include the following operators: **!** for **not**, **&** for **and, |** for **or**, the use of parenthesis for operations grouping is also allowed.
To perform the evaluation of an expression, it will be considered the priority of the operators, the **not** having the highest, and the **or** the lowest. The program must yield **V** or **F**, as the result for each expression in the input file.

Input

The expressions are of a variable length, although will never exceed 100 symbols. Symbols may be separated by any number of spaces or no spaces at all, therefore, the total length of an expression, as a number of characters, is unknown.
The number of expressions in the input file is variable and will never be greater than 20. Each expression is presented in a new line, as shown below.

Output

For each test expression, print "Expression " followed by its sequence number, ": ", and the resulting value of the corresponding test expression. Separate the output for consecutive test expressions with a new line.
Use the same format as that shown in the sample output shown below.

Input Example:

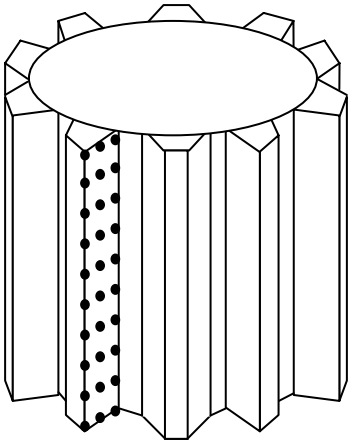
```
( V | V ) & F & ( F| V)
!V | V & V & !F & (F | V ) & (!F | F | !V & V)
(F&F|V|!V&!F&!(F|F&V))
```

Output Example:

```
Expression 1: F
Expression 2: V
Expression 3: V
```

ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 3 – Gear Profiles
Source Program: GearP.cpp, GearP.java, GearP.c, GearP.pas
Input file: GearP.in **Output file**: GearP.out

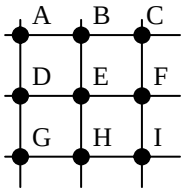


The Dynamic Gears Company is developing a new program to manufacture gears based on a master gear. The first step consists on making measurements of the master gear using the T-500 machine. This machine creates a file with coordinates of different points on one of the gear's tooth.

The T-500 machine finds the coordinates of several points on the surface of one of the tooth's face. These points represent a grid that defines the entire surface (as seen on the image). The T-500 machine measures *X* points on the tooth from top to bottom, then it moves towards the inside to read again *X* points. This sweeping is done *Y* times until the gear is complete. The final result of the T-500 machine is a file with a consecutive list of $X \cdot Y$ points.

The Gear Technology Department has to develop a program capable of calculating the geometry of the tooth based on the grid of points, and generate the instructions for the machine that will cut the gears. After analyzing the problem it was decided that the Normal value of several points has to be calculated so that the cutting machine knows how to generate the gear.

To be able to calculate the Normal value, 9 points from the grid have to be considered together (as seen on the diagram). For example, if the T-500 machine read 5 points from top to bottom and 3 columns from left to right, points A, D, and G come from the first three points the T-500 machine read. Points B, E, and H come from points 6, 7, and 8. Points C, F, and I come from points 11, 12, and 13. The small 9-points grid contains the required values to calculate the Normal value using the following formula:



$$\mathrm{Norm}_x = N_x \cdot Factor$$
$$Norm_y = N_y \cdot Factor$$
$$Norm_z = N_z \cdot Factor$$

Where:

$$Factor = \frac{1}{\sqrt{N_x^2 + N_y^2 + N_z^2}}$$

$$N_x = H\tau_y \cdot V\tau_z - H\tau_z \cdot V\tau_y$$
$$N_y = H\tau_z \cdot V\tau_x - H\tau_x \cdot V\tau_z$$
$$N_z = H\tau_x \cdot V\tau_y - H\tau_y \cdot V\tau_x$$

$$H\tau_x = ((F_y - E_y) \cdot HF_z - (F_z - E_z) \cdot HF_y) - (F_x - E_x) \cdot Hu$$
$$H\tau_y = ((F_z - E_z) \cdot HF_x - (F_x - E_x) \cdot HF_z) - (F_y - E_y) \cdot Hu$$
$$H\tau_z = ((F_x - E_x) \cdot HF_y - (F_y - E_y) \cdot HF_x) - (F_z - E_z) \cdot Hu$$

$$V\tau_x = ((H_y - E_y) \cdot VF_z - (H_z - E_z) \cdot VF_y) - (H_x - E_x) \cdot Vu$$
$$V\tau_y = ((H_z - E_z) \cdot VF_x - (H_x - E_x) \cdot VF_z) - (H_y - E_y) \cdot Vu$$
$$V\tau_z = ((H_x - E_x) \cdot VF_y - (H_y - E_y) \cdot VF_x) - (H_z - E_z) \cdot Vu$$

$$Hu = (E_x - D_x) \cdot (D_x - F_x) + (E_y - D_y) \cdot (D_y - F_y) + (E_z - D_z) \cdot (D_z - F_z)$$
$$Vu = (E_x - B_x) \cdot (B_x - H_x) + (E_y - B_y) \cdot (B_y - H_y) + (E_z - B_z) \cdot (B_z - H_z)$$

$$HF_x = (D_y - F_y) \cdot (E_z - D_Z) - (D_z - F_z) \cdot (E_y - D_y)$$
$$HF_y = (D_z - F_z) \cdot (E_x - D_x) - (D_x - F_x) \cdot (E_z - D_z)$$
$$HF_z = (D_x - F_x) \cdot (E_y - D_y) - (D_y - F_y) \cdot (E_x - D_x)$$

$$VF_x = (B_y - H_y) \cdot (E_z - B_Z) - (B_z - H_z) \cdot (E_y - B_y)$$
$$VF_y = (B_z - H_z) \cdot (E_x - B_x) - (B_x - H_x) \cdot (E_z - B_z)$$
$$VF_z = (B_x - H_x) \cdot (E_y - B_y) - (B_y - H_y) \cdot (E_x - B_x)$$

The program to develop must go through all points on the grid that the T-500 machine generated, using the 9-points grid, moving from top to bottom first, and then from left to right. The first 9-points grid, therefore, uses points A=1, D=2, G=3, B=6, E=7, H=8, C=11, F=12, and I=13; the second 9-points grid uses points A=2, D=3, G=4, B=7, E=8, H=9, C=12, F=13, and I=14; etc.

Input

The input represents the file generated by the T-500 machine with only one case. The first line contains two positive integers *X* (3≤*X*≤50) and *Y* (3≤*Y*≤50). *X* is the number of points read from top to bottom, and *Y* is the number of columns read from left to right.

All the following lines have 3 floating-point numbers: *x*, *y*, and *z*. The three values represent the coordinates of the measured point in the Cartesian plane. Each number has a maximum of 4 decimals. The *x* values starts at character 1, the *y* value at character 15, and the *z* value at character 30.

Output

The output must contain on the first line the total number of points contained on the input. Starting on the second line, the $Norm_x$, $Norm_y$ and $Norm_z$ values from all calculated Normal values must be printed, one line for each calculation. The values must be floating-point numbers with a maximum of 4 decimals. The $Norm_x$ value must start on the 1st column, the $Norm_y$ value on the column 15, and the $Norm_z$ value on the column 30.

Input Example

```
5 3
-36.9469    225.1665    150
-36.7819    226.3999    150
-36.5822    227.7063    150
-36.3451    229.0841    150
-36.0678    230.5327    150
-35.7464    232.0432    150
-34.3768    237.1574    150
-33.9837    238.7979    150
-33.5259    240.2806    150
-33.0456    241.757     150
-32.5119    243.3183    150
-31.9384    244.9173    150
-32.8032    225.8074    135
-32.6156    227.0375    135
-32.392     228.34      135
```

Output Example

```
15
0.          0.          1.
0.9423      -0.2594     0.2117
0.9373      -0.2972     0.1819
```

ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 4 – Lazy Pianist
Source Program: Pianist.cpp, Pianist.java, Pianist.c, Pianist.pas
Input file: Pianist.in **Output file**: Pianist.out

Music and mathematics always had a close relationship. Since Pythagoras, it is known that tonal harmony is closely related to the numerical relation of the frequencies.

A lazy pianist, that uses to play VEERY LAARGE melodies composed by Kindermann, has taken advantage of this narrow relationship. However, this pianist commonly plays only the half or a quarter of the total notes. Amazingly, the audience never has realized his trick. The main reason is that this pianist always plays melodies that have a particular characteristic that is better described by the sequence of melody shown in figure 1.

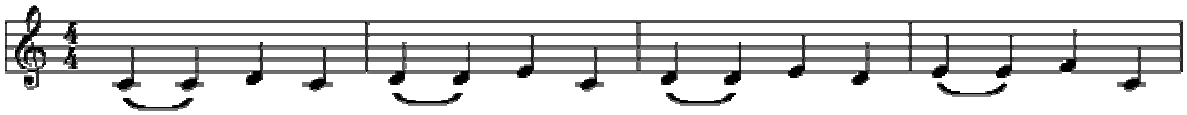


Figure 1. A sequence generated by Kindermann

As you can see, the complete sequence is:
DO DO RE DO RE RE MI DO RE RE MI RE MI MI FA DO

However, if we choose only even notes, we can obtain the following sequence:
DO **DO** RE **DO** RE **RE** MI **DO** RE **RE** MI **RE** MI **MI** FA **DO**
DO DO RE DO RE RE MI DO

This is exactly the same melody. Even more, we can observe that choosing only every fourth note, we obtain:
DO DO RE …
And, this is exactly the same melody.

The lazy pianist would like to continue playing this kind of melodies, but he does not know how to complete the melody shown in figure 1. You must construct an algorithm in order to complete that melody.

A useful tip to do the job would be to assume that every note should be represented by an integer. Thus, DO=1, RE=2, MI=3, etc. Thus, sequence presented earlier could be seen as:

1
1 **1 2**
1 1 2 **1 2 2 3**
1 1 2 1 2 2 3 **1 2 2 3 2 3 3 4**

ó

1
1 1 **2**
1 1 **2** 1 **2** 2 **3**
1 1 **2** 1 **2** 2 **3** 1 **2** 2 **3** 2 **3** 3 **4**

A possible algorithm for generating the sequence is shown in table 1. Every decimal number is expressed by its binary representation, and then we add the number of 1's that exist in that representation. This value indicates the note.

decimal number	binary number	digits sum	Tone	2nd	4th
1	**1**	1	DO		
2	**1**0	1	DO	DO	
3	**11**	2	RE	.	
4	**1**00	1	DO	DO	DO
5	**101**	2	RE	.	.
6	**11**0	2	RE	RE	.
7	**111**	3	MI	.	.
8	**1**000	1	DO	DO	DO
9	**1001**	2	RE	.	.
10	**101**0	2	RE	RE	.
11	**1011**	3	MI	.	.
12	**11**00	2	RE	RE	RE
13	**1101**	3	MI	.	.
14	**111**0	3	MI	MI	.
15	**1111**	4	FA	.	.
16	**1**0000	1	DO	DO	DO
...	...	...	...	...	...

Table 1. Tip for generating the sequence.

Input

Input consists of a list of three positive integers (total, cant, pos), preceded by an integer number that means the total of cases. Where total means total of tones to generate, cant means how many tones should be shown as a result, and pos means the start point. In this way, if cant=2, and pos=1, then this mean that it is needed to generate a sequence of 2 values starting from position number one. It is not allowed that pos has a value greater than total, and obviously neither that
((pos + cant) -1) > total.

Output

A sequence of positive integers separated by a space. Each integer means one note of the melody.

Input Example

5
16 3 10
2 2 1
5 5 2
8 1 8
15 15 1

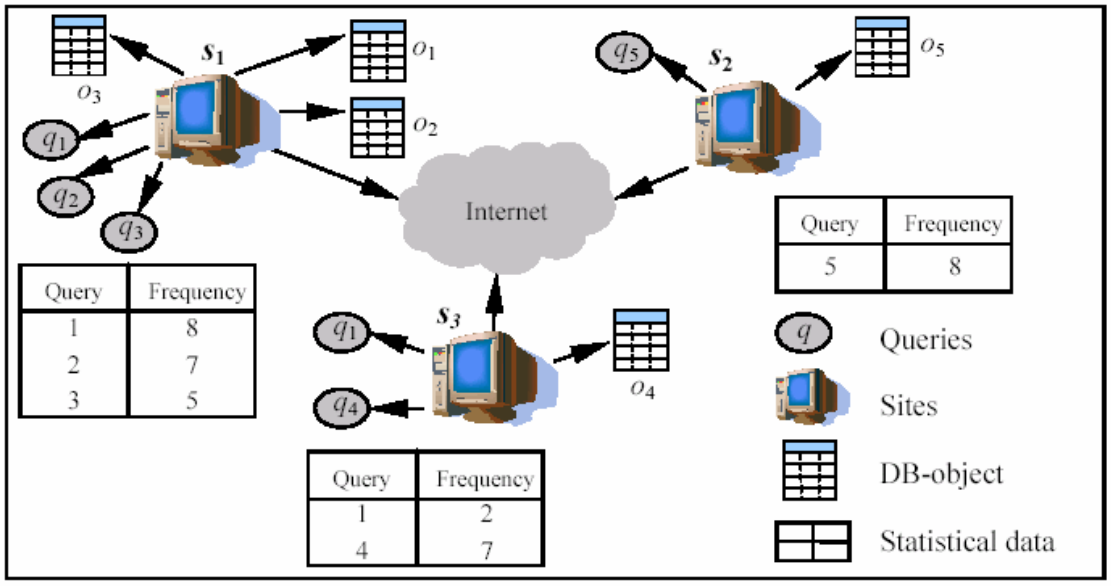
Output Example

Case 1: 2 3 2
Case 2: 1 1
Case 3: It cannot be solved.
Case 4: 1
Case 5: 1 1 2 1 2 2 3 1 2 2 3 2 3 3 4

ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 5 – Grouping Tuples
Source Program: GTuples.cpp, GTuples.java, GTuples.c, GTuples.pas
Input file: GTuples.in **Output file**: GTuples.out

The DDB distribution design problem consists of allocating DB-objects, such that the total cost of data transmission for processing all the applications is minimized. A DB-object is an entity of a database, which can be an attribute, a tuples set, a relation or a file. DB-objects are independent units that must be allocated in the sites of a network. A formal definition of the problem is the following:



Distribution design problem

Let us consider a set of DB-objects $O = \{o_1, o_2, ..., o_{no}\}$, a computer communication network that consists of a set of sites $S = \{s_1, s_2, ..., s_{ns}\}$, where a set of queries $Q = \{q_1, q_2, ..., q_{nq}\}$ are executed, the DB-objects required by each query, an initial DB-object allocation schema, and the access frequencies of each query from each site in a time period. The problem consists of obtaining a new replicated allocation schema that adapts to a new database usage pattern and minimizes transmission costs. The figure shows the main elements related with this problem.

The distribution design problem consists of determining the location of data so that the access and communication costs are minimized. Like many other real problems, it is a combinatorial NP-Hard problem. Some common situation that we can observe when you solve very large instances of an NP-Hard problem are: large requirements of time and large demands of memory resources. One approach proposed to solve this kind of problems consist of applying a grouping methodology.

For *nt* tuples and *ns* sites, the method consist in having the access frequency matrix in the sites F[*nt*][*ns*] of the objective instance, transform it in an access patterns matrix and to group the tuples with the same pattern.

The pattern of access for a tuple is the binary vector indicating from which site the tuple is accessed.
For example:
From the next access frequency matrix:

$$\begin{pmatrix} 2 & 3 & 0 & 4 \\ 0 & 0 & 6 & 3 \\ 3 & 4 & 0 & 2 \\ 1 & 0 & 1 & 5 \\ 0 & 0 & 3 & 1 \end{pmatrix}$$

Then corresponding access patterns matrix is:

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

It is observed than the rows 1 and 3 have the same access patterns, so the rows 2 and 5. Then it will be three groups considering the third group formed by the single row 4. The cardinality of a group is defined by the number of tuples included in it.

The reduced access frequency matrix obtained by the sum of the tuples in the group will be:

$$\begin{pmatrix} 5 & 7 & 0 & 6 \\ 0 & 0 & 9 & 4 \\ 1 & 0 & 1 & 5 \end{pmatrix}$$

You must design a program that accept n matrices of access frequency and for each one get the groups formed by the tuples with the same pattern of access. Also you must get the reduced access frequency matrix.

Input
The input will consist of a series of problems separated by a blank row. Each problem will contain in the first line two integers indicating the dimension of the access frequency matrix F[*nt*][*ns*], where $2 \leq nt \leq 50{,}000$ and $3 \leq ns \leq 15$, the integers are separated by a blank. In the next *nt* consecutive lines appear the *ns* integers of the access frequency matrix separated by a blank space.

Output
The solutions of the different problems are separated by a blank line.
For each problem in the input the output will contain:
The first line will contain two integers indicating the number of rows and columns of the reduced frequency matrix, follows by the items of the reduced frequency matrix, a row by line,

the next is a blank line, followed by a set of consecutive lines containing pair of integers which represent the group number and the group cardinality ordered by the first occurrence of the pattern.

Input Example

```
15 4
3 0 1 2
0 1 1 5
5 1 0 0
2 0 5 1
0 2 2 4
1 1 0 4
8 9 10 0
7 2 0 0
9 7 0 0
0 0 4 0
8 0 3 1
5 0 7 9
1 0 1 1
0 0 0 0
1 1 1 1

20 3
0 3 0
0 0 6
3 0 0
0 0 4
0 0 6
0 0 3
0 0 2
3 0 0
0 0 3
0 0 3
6 0 0
3 0 0
0 0 2
0 3 0
0 0 3
0 3 0
3 0 0
0 1 0
0 0 3
0 3 0
```

Output Example

```
8 4
19 0 17 14
0 3 3 9
```

21 10 0 0
1 1 0 4
8 9 10 0
0 0 4 0
0 0 0 0
1 1 1 1

1 5
2 2
3 3
6 1
7 1
10 1
14 1
15 1

3 3
0 13 0
0 0 35
18 0 0

1 5
2 10
3 5

ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 6 – Power of Cryptography
Source Program: Crypto.cpp, Crypto.java, Crypto.c, Crypto.pas
Input file: Crypto.in **Output file**: Crypto.out

Current work in cryptography involves (among other things) large prime numbers and computing powers of numbers among these primes. Work in this area has resulted in the practical use of results from number theory and other branches of mathematics once considered to be only of theoretical interest.

This problem involves the efficient computation of integer roots of numbers.

Problem

Given an integer n>=1 and an integer p>= 1 you have to write a program that determines the n^{th} positive root of p. In this problem, given such integers n and p, p will always be of the form k to the nth. power, for an integer k (this integer is what your program must find).

Input

The input consists of a sequence of integer pairs n and p with each integer on a line by itself. For all such pairs 1<=n<= 200, 1<=p<10^{101} and there exists an integer k, 1<=k<=10^{9} such that $k^n = p$.

Output

For each integer pair n and p the value k should be printed, i.e., the number k such that $k^n=p$.

Input Example

2 16
3 27
7 4357186184021382204544

Output Example

4
3
1234

ACM International Collegiate Programming Contest 2004
México and Central America Region

Problem No. 7 – Factors in Bin-packing Instances
Source Program: Bin.cpp, Bin.java, Bin.c, Bin.pas
Input file: Bin.in **Output file**: Bin.out

Bin-packing is a classic computer problem that is defined with the parameters c, n, and s_i. In this, there is a given sequence of n items each one with a given size $0 < s_i \leq c$, and an unlimited number of bins each one of capacity c. The question is to determine the smallest number of bins into which the objects can be packed. Two real examples of this problem type are: 1) elaboration of wooden bookcases by making cuts of different lengths from boards of a fixed length, 2) packed of advertisements of different lengths into fixed length commercial breaks on radio or television. Due to Bin-packing complexity, algorithms of approximate solution (meta heuristics) have been developed. In order to analyze the performance of these algorithms, it is needed to generate Bin-packing instances with properties that affect to the algorithms in different ways. With this purpose, the parameter values of the instances are obtained in a random way, and later on they are adjusted to get the wanted property. The property of easy-packing is determined mainly with the index of factors.

Index of factors f: expresses the proportion of items whose sizes are factors of the bin capacity; in where is understood that an item i is a factor when its size s_i divides from an exact way to the bin capacity c. In general, instances with many factors are considered easy to solve.

$$f = \frac{\sum_{i=1}^{n} factor(c, s_i)}{n}$$

where:

$$factor(c, s_i) = \begin{cases} 1 & \text{if } (c \bmod s_i) = 0 \\ 0 & \text{otherwhise} \end{cases}$$

Develop a program, that given the parameter values c, n and s_i for defining a Bin-packing instance, and given a value for the index of factors f, modify the given instance so that its easy-packing property acquires the value specified by f. The modification must be carried out under the next conditions:

a) Suppose that in the instance given, the number of items that are exact divisors of c, is always smaller than the number indicated by $\lceil f \times n \rceil$. Remember that the function $\lceil\ \rceil$ rounds to the superior integer.

b) Modify sizes of items, that are not exact divisors, to complete the number of factors indicated by $\lceil f \times n \rceil$. Select the items that require smaller adjustment. In matching case, assign bigger priority to the items located in a position smaller.

Input
In the first line appears a value among 0 and 1 corresponding to the index of factors *f*. In the second line appears the number of items *n* and the bin capacity *c*; they are separated by coma. The sizes of the *n* objects appear in consecutive lines, each size in a line.

Output
The result is the instance modified in the number of factors. In the first line appears the number of items *n* and the bin capacity *c*; they are separated by coma. The sizes of the *n* objects appear in consecutive lines, each size in a line.

Input Example
0.6
5, 50
39
43
10
18
6

Output Example
5, 50
39
50
10
18
5