










acm international collegiate programming contest

ACM International Collegiate Programming Contest
Sponsored by IBM

Central Europe 2000 Regional Contest

Czech Technical University in Prague
Dept. of Computer Science, Faculty of Electrical Engineering

November 11, 2000

-
- | | |
|---|---|
|  | Simple Arithmetics |
|  | The Bulk |
|  | Complete the sequence! |
|  | Direct Visibility |
|  | Complicated Expressions |
|  | Factorial |
|  | The Game of Master-Mind |
|  | Hotline |
|  | I-Keyboard |

Introduction

Dear contestants:

As usual, we have prepared a set of real-life problems for you. This time, we are interested in the mobile communications which are becoming very popular. A new company appeared on this market recently. It has French name because French is an official language of communications. The name is *Association de Communication Mobile (ACM)* and perhaps all of you know enough French to understand it.

The company plans to become a market leader so it makes a lot of activities. Beside its own GSM network, it also creates a new WAP portal to serve as a comprehensive information

source for customers having phones with WAP browsers. Moreover, ACM also wants to produce its own cellular phone with several smart features. Obviously, creating so many new things, the company also faces many interesting and complex problems. Most of them can be solved with the help of computers. Your task is to write a set of computer programs to solve some of the most appealing problems.

All of the programs will run in UNIX environment. Every program should read the data from the standard input and send results to the standard output. The data format is given and must be exactly followed. There will be no extra spaces in the input and there must be no extra spaces and newlines in the output, if not stated otherwise. If the description of a particular problem does not say anything else, all numbers will fit into standard signed *integer* type that has 32 bits.

We wish you a lot of fun and good luck with solving the problems from the communication technology area.

Your organising team



Simple Arithmetics

One part of the new WAP portal is also a calculator computing expressions with very long numbers. To make the output look better, the result is formatted the same way as is it usually used with manual calculations.

Your task is to write the core part of this calculator. Given two numbers and the requested operation, you are to compute the result and print it in the form specified below. With addition and subtraction, the numbers are written below each other. Multiplication is a little bit more complex: first of all, we make a partial result for every digit of one of the numbers, and then sum the results together.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of expressions to follow. Each expression consists of a single line containing a positive integer number, an operator (one of +, - and *) and the second positive integer number. Every number has at most 500 digits. There are no spaces on the line. If the operation is subtraction, the second number is always lower than the first one. No number will begin with zero.

Output Specification

For each expression, print two lines with two given numbers, the second number below the first one, last digits (representing unities) must be aligned in the same column. Put the operator right in front of the first digit of the second number. After the second number, there must be a horizontal line made of dashes (-).

For each addition or subtraction, put the result right below the horizontal line, with last digit

aligned to the last digit of both operands.

For each multiplication, multiply the first number by each digit of the second number. Put the partial results one below the other, starting with the product of the last digit of the second number. Each partial result should be aligned with the corresponding digit. That means the last digit of the partial product must be in the same column as the digit of the second number. No product may begin with any additional zeros. If a particular digit is zero, the product has exactly one digit -- zero. If the second number has more than one digit, print another horizontal line under the partial results, and then print the sum of them.

There must be minimal number of spaces on the beginning of lines, with respect to other constraints. The horizontal line is always as long as necessary to reach the left and right end of both numbers (and operators) right below and above it. That means it begins in the same column where the leftmost digit or operator of that two lines (one below and one above) is. It ends in the column where is the rightmost digit of that two numbers. The line can be neither longer nor shorter than specified.

Print one blank line after each test case, including the last one.

Sample Input

```
4
12345+67890
324-111
325*4405
1234*4
```

Sample Output

```
 12345
+67890
-----
 80235
```

```
  324
-111
----
  213
```

```
   325
*4405
-----
  1625
   0
 1300
1300
-----
1431625
```

```
1234
 *4
----
4936
```

The Bulk

ACM uses a new special technology of building its transceiver stations. This technology is called *Modular Cuboid Architecture (MCA)* and is covered by a patent of Lego company. All parts of the transceiver are shipped in unit blocks that have the form of cubes of exactly the same size. The cubes can be then connected to each other. The MCA is modular architecture, that means we can select preferred transceiver configuration and buy only those components we need.

The cubes must be always connected "face-to-face", i.e. the whole side of one cube is connected to the whole side of another cube. One cube can be thus connected to at most six other units. The resulting equipment, consisting of unit cubes is called *The Bulk* in the communication technology slang.

Sometimes, an old and unneeded bulk is condemned, put into a storage place, and replaced with a new one. It was recently found that ACM has many of such old bulks that just occupy space and are no longer needed. The director has decided that all such bulks must be disassembled to single pieces to save some space. Unfortunately, there is no documentation for the old bulks and nobody knows the exact number of pieces that form them. You are to write a computer program that takes the bulk description and computes the number of unit cubes.

Each bulk is described by its faces (sides). A special X-ray based machine was constructed that is able to localise all faces of the bulk in the space, even the inner faces, because the bulk can be partially hollow (it can contain empty spaces inside). But any bulk must be connected (i.e. it cannot drop into two pieces) and composed of whole unit cubes.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of bulks to follow. Each bulk description begins with a line containing single positive integer F , $6 \leq F \leq 250$, stating the number of faces. Then there are F lines, each containing one face description. All faces of the bulk are always listed, in any order. Any face may be divided into several distinct parts and described like if it was more faces. Faces do not overlap. Every face has one inner side and one outer side. No side can be "partially inner and partially outer".

Each face is described on a single line. The line begins with an integer number P stating the number of points that determine the face, $4 \leq P \leq 200$. Then there are $3 \times P$ numbers, coordinates of the points. Each point is described by three coordinates X, Y, Z ($0 \leq X, Y, Z \leq 1000$) separated by spaces. The points are separated from each other and from the number P by two space characters. These additional spaces were added to make the input more human readable. The face can be constructed by connecting the points in the specified order, plus connecting the last point with the first one.

The face is always composed of "unit squares", that means every edge runs either in X , Y or

Z-axis direction. If we take any two neighbouring points X_1, Y_1, Z_1 and X_2, Y_2, Z_2 , then the points will always differ in exactly one of the three coordinates. I.e. it is either $X_1 \neq X_2$, or $Y_1 \neq Y_2$, or $Z_1 \neq Z_2$, other two coordinates are the same. Every face lies in an orthogonal plane, i.e. exactly one coordinate is always the same for all points of the face. The face outline will never touch nor cross itself.

Output Specification

Your program must print a single line for every test case. The line must contain the sentence The bulk is composed of V units., where V is the volume of the bulk.

Sample Input

```

2
12
4 10 10 10 10 10 20 10 20 20 10 20 10
4 20 10 10 20 10 20 20 20 20 20 10
4 10 10 10 10 10 20 20 10 20 10 10
4 10 20 10 10 20 20 20 20 20 20 10
4 10 10 10 10 20 10 20 20 10 20 10 10
5 10 10 20 10 20 20 20 20 20 20 15 20 20 10 20
4 14 14 14 14 14 16 14 16 16 14 16 14
4 16 14 14 16 14 16 16 16 16 16 14
4 14 14 14 14 14 16 16 14 16 16 14 14
4 14 16 14 14 16 16 16 16 16 16 14
4 14 14 14 14 16 14 16 16 14 16 14 14
4 14 14 16 14 16 16 16 16 16 14 16
12
4 20 20 30 20 30 30 30 30 30 30 20 30
4 10 10 10 10 40 10 40 40 10 40 10 10
6 10 10 20 20 10 20 20 30 20 30 30 20 30 40 20 10 40 20
6 20 10 20 20 20 20 30 20 20 30 40 20 40 40 20 40 10 20
4 10 10 10 40 10 10 40 10 20 10 10 20
4 10 40 10 40 40 10 40 40 20 10 40 20
4 20 20 20 30 20 20 30 20 30 20 20 30
4 20 30 20 30 30 20 30 30 30 20 30 30
4 10 10 10 10 40 10 10 40 20 10 10 20
4 40 10 10 40 40 10 40 40 20 40 10 20
4 20 20 20 20 30 20 20 30 30 20 20 30
4 30 20 20 30 30 20 30 30 30 30 20 30

```

Sample Output

```

The bulk is composed of 992 units.
The bulk is composed of 10000 units.

```

Complete the sequence!

You probably know those quizzes in Sunday magazines: given the sequence 1, 2, 3, 4, 5, what is the next number? Sometimes it is very easy to answer, sometimes it could be pretty

hard. Because these "sequence problems" are very popular, ACM wants to implement them into the "Free Time" section of their new WAP portal.

ACM programmers have noticed that some of the quizzes can be solved by describing the sequence by polynomials. For example, the sequence 1, 2, 3, 4, 5 can be easily understood as a trivial polynomial. The next number is 6. But even more complex sequences, like 1, 2, 4, 7, 11, can be described by a polynomial. In this case, $1/2 \cdot n^2 - 1/2 \cdot n + 1$ can be used. Note that even if the members of the sequence are integers, polynomial coefficients may be any real numbers.

Polynomial is an expression in the following form:

$$P(n) = a_D \cdot n^D + a_{D-1} \cdot n^{D-1} + \dots + a_1 \cdot n + a_0$$

. If $a_D \neq 0$, the number D is called a *degree* of the polynomial. Note that constant function $P(n) = C$ can be considered as polynomial of degree 0, and the zero function $P(n) = 0$ is usually defined to have degree -1.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of test cases to follow. Each test case consists of two lines. First line of each test case contains two integer numbers S and C separated by a single space, $1 \leq S < 100$, $1 \leq C < 100$, $(S+C) \leq 100$. The first number, S , stands for the length of the given sequence, the second number, C is the amount of numbers you are to find to complete the sequence.

The second line of each test case contains S integer numbers X_1, X_2, \dots, X_S separated by a space. These numbers form the given sequence. The sequence can always be described by a polynomial $P(n)$ such that for every i , $X_i = P(i)$. Among these polynomials, we can find the polynomial P_{min} with the lowest possible degree. This polynomial should be used for completing the sequence.

Output Specification

For every test case, your program must print a single line containing C integer numbers, separated by a space. These numbers are the values completing the sequence according to the polynomial of the lowest possible degree. In other words, you are to print values $P_{min}(S+1)$, $P_{min}(S+2)$, ..., $P_{min}(S+C)$.

It is guaranteed that the results $P_{min}(S+i)$ will be non-negative and will fit into the standard *integer* type.

Sample Input

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

```

8 2
1 2 4 7 11 16 22 29
10 2
1 1 1 1 1 1 1 1 2
1 10
3

```

Sample Output

```

7 8 9
37 46
11 56
3 3 3 3 3 3 3 3 3

```

Direct Visibility

Building the GSM network is a very expensive and complex task. Moreover, after the *Base Transceiver Stations (BTS)* are built and working, we need to perform many various measurements to determine the state of the network, and propose effective improvements to be made.

The ACM technicians have a special equipment for measuring the strength of electromagnetic fields, the transceivers' power and quality of the signal. This equipment is packed into a huge knapsack and the technician must move with it from one BTS to another. Unfortunately, the knapsack have not enough memory for storing all of the measured values. It has a small cache only, that can store values for several seconds. Then the values must be transmitted to the BTS by an infrared connection (IRDA). The IRDA needs direct visibility between the technician and the BTS.

Your task is to find the path between two neighbouring BTSes such that at least one of those BTSes is always visible.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of test cases to follow. Each test case consists of a town description. For simplicity, a town is modelled as a rectangular grid of $P \times Q$ square fields. Each field is exactly 1 metre wide. For each field, a non-negative integer $Z_{i,j}$ is given, representing the height of the terrain in that place, in metres. That means the town model is made of cubes, each of them being either solid or empty. There are no "half solid" cubes.

The first line of each test case contains two integer numbers P and Q , separated by a single space, $1 \leq P, Q \leq 200$. Then there are P lines each containing Q integer numbers separated by a space. These numbers are $Z_{i,j}$, where $1 \leq i \leq P$, $1 \leq j \leq Q$ and $0 \leq Z_{i,j} \leq 5000$. After the terrain description, there are four numbers R_1, C_1, R_2, C_2 on the last line of each test case. These numbers represent position of two BTSes, $1 \leq R_1, R_2 \leq P$, $1 \leq C_1, C_2 \leq Q$. The first coordinate (R) determines the row of the town, the second coordinate determines

the column.

The technician is moving in steps (*steps* stands for *Standard Technician's Elementary Positional Shift*). Each step is made between two neighbouring square fields. That means the step is always in North, South, West or East direction. It is not possible to move diagonally. The step between two fields *A* and *B* (step from *A* to *B*) is allowed only if the height of the terrain in the field *B* is not very different from the height in the field *A*. The technician can climb at most 1 metre up or descend at most 3 metres down in a single step.

At the end of each step, at least one of the two BTSes must be visible. However, there can be some point "in the middle of the step" where no BTS is visible. This is OK and the data is handled by the cache. The BTS is considered visible, if there is a direct visibility between the unit cube just above the terrain on the BTSes coordinates and the cube just above the terrain on the square field, where the technician is. Direct visibility between two cubes means that the line connecting the centres of the two cubes does not intersect any solid cube. However, the line can touch any number of solid cubes. In other words, consider both the BTS and the technician being points exactly half metre above the surface and in the centre of the appropriate square field.

Note that the IRDA beam can go between two cubes that touch each other by their edge, although there is no space between them. It is because such a beam touches both of these two cubes but does not intersect any of them. See the last test case of the sample input for an example of such a situation.

Output Specification

You are to find the shortest possible path meeting the above criteria. All steps must be done between neighbouring fields, the terrain must not elevate or descend too much, and at the end of each step, at least one BTS must be visible.

For each test case, print one line containing the sentence `The shortest path is M steps long.`, where M is the number of steps that must be made. If there is no such path, output the sentence `Mission impossible!`.

Sample Input

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

```

2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
9 9 9 9 9 9 9 2
2 2 2 2 2 2 2 2
1 5 5 1
6 12
5 5 5 5 1 5 5 5 5 5 5 5
5 5 5 5 1 5 5 5 5 5 5 5
5 5 5 5 9 5 5 5 5 5 5 5
5 9 1 5 5 5 5 5 5 5 5 5
5 5 9 5 5 5 5 5 5 5 5 5
5 5 9 5 5 5 5 5 5 5 5 5
6 1 3 12

```

Sample Output

```

The shortest path is 10 steps long.
Mission impossible!
The shortest path is 14 steps long.
The shortest path is 18 steps long.

```

Complicated Expressions

The most important activity of ACM is the GSM network. As the mobile phone operator, ACM must build its own transmitting stations. It is very important to compute the exact behaviour of electro-magnetic waves. Unfortunately, prediction of electro-magnetic fields is a very complex task and the formulas describing them are very long and hard-to-read. For example, below are the Maxwell's Equations describing the basic laws of electrical engineering.

$$\nabla \vec{B} = 0 \qquad \nabla \vec{E} = \frac{\rho}{\epsilon_0}$$

$$\nabla \times \vec{H} = \vec{j} + \frac{\partial \vec{D}}{\partial t} \qquad \nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}$$

ACM has designed its own computer system that can make some field computations and produce results in the form of mathematic expressions. Unfortunately, by generating the expression in several steps, there are always some unneeded parentheses inside the expression. Your task is to take these partial results and make them "nice" by removing all unnecessary parentheses.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of expressions to follow. Each expression consists of a single line containing only lowercase letters, operators (+, -, *, /) and parentheses ((and)). The letters are variables that can have any value, operators and parentheses have their usual meaning. Multiplication and division have higher priority than subtraction and addition. All operations with the same priority are computed from left to right (operators are left-associative). There are no spaces inside the expressions. No input line contains more than 250 characters.

Output Specification

Print a single line for every expression. The line must contain the same expression with unneeded parentheses removed. You must remove as many parentheses as possible without changing the semantics of the expression. The semantics of the expression is considered the same if and only if any of the following conditions hold:

- The ordering of operations remains the same. That means " $(a+b)+c$ " is the same as " $a+b+c$ ", and " $a+(b/c)$ " is the same as " $a+b/c$ ".
- The order of some operations is swapped but the result remains unchanged with respect to the addition and multiplication associativity. That means " $a+(b+c)$ " and " $(a+b)+c$ " are the same. We can also combine addition with subtraction and multiplication with division, if the subtraction or division is the second operation. For example, " $a+(b-c)$ " is the same as " $a+b-c$ ".

You cannot use any other laws, namely you cannot swap left and right operands and you cannot replace " $a-(b-c)$ " with " $a-b+c$ ".

Sample Input

```
8
(a+(b*c))
((a+b)*c)
(a*(b*c))
(a*(b/c)*d)
((a/(b/c))/d)
((x))
(a+b)-(c-d)-(e/f)
(a+b)+(c-d)-(e+f)
```

Sample Output

```
a+b*c
(a+b)*c
a*b*c
a*b/c*d
a/(b/c)/d
x
a+b-(c-d)-e/f
a+b+c-d-(e+f)
```

Factorial

The most important part of a GSM network is so called *Base Transceiver Station (BTS)*. These transceivers form the areas called *cells* (this term gave the name to the cellular phone) and every phone connects to the BTS with the strongest signal (in a little simplified view). Of course, BTSes need some attention and technicians need to check their function periodically.

ACM technicians faced a very interesting problem recently. Given a set of BTSes to visit,

they needed to find the shortest path to visit all of the given points and return back to the central company building. Programmers have spent several months studying this problem but with no results. They were unable to find the solution fast enough. After a long time, one of the programmers found this problem in a conference article. Unfortunately, he found that the problem is so called "Travelling Salesman Problem" and it is very hard to solve. If we have N BTSes to be visited, we can visit them in any order, giving us $N!$ possibilities to examine. The function expressing that number is called factorial and can be computed as a product $1.2.3.4....N$. The number is very high even for a relatively small N .

The programmers understood they had no chance to solve the problem. But because they have already received the research grant from the government, they needed to continue with their studies and produce at least *some* results. So they started to study behaviour of the factorial function.

For example, they defined the function Z . For any positive integer N , $Z(N)$ is the number of zeros at the end of the decimal form of number $N!$. They noticed that this function never decreases. If we have two numbers $N_1 < N_2$, then $Z(N_1) \leq Z(N_2)$. It is because we can never "lose" any trailing zero by multiplying by any positive number. We can only get new and new zeros. The function Z is very interesting, so we need a computer program that can determine its value efficiently.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of numbers to follow. Then there is T lines, each containing exactly one positive integer number N , $1 \leq N \leq 1000000000$.

Output Specification

For every number N , output a single line containing the single non-negative integer $Z(N)$.

Sample Input

```
6
3
60
100
1024
23456
8735373
```

Sample Output

```
0
14
24
253
5861
2183837
```

The Game of Master-Mind

If you want to buy a new cellular phone, there are many various types to choose from. To decide which one is the best for you, you have to consider several important things: its size and weight, battery capacity, WAP support, colour, price. One of the most important things is also the list of games the phone provides. Nokia is one of the most successful phone makers because of its famous Snake and Snake II. ACM wants to make and sell its own phone and they need to program several games for it. One of them is Master-Mind, the famous board logical game.

The game is played between two players. One of them chooses a *secret code* consisting of P ordered pins, each of them having one of the predefined set of C colours. The goal of the second player is to guess that secret sequence of colours. Some colours may not appear in the code, some colours may appear more than once.

The player makes guesses, which are formed in the same way as the secret code. After each guess, he/she is provided with an information on how successful the guess was. This feedback is called a *hint*. Each hint consists of B black points and W white points. The black point stands for every pin that was guessed right, i.e. the right colour was put on the right position. The white point means right colour but on the wrong position. For example, if the secret code is "white, yellow, red, blue, white" and the guess was "white, red, white, white, blue", the hint would consist of one black point (for the white on the first position) and three white points (for the other white, red and blue colours). The goal is to guess the sequence with the minimal number of hints.

The new ACM phone should have the possibility to play both roles. It can make the secret code and give hints, but it can also make its own guesses. Your goal is to write a program for the latter case, that means a program that makes Master-Mind guesses.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of test cases to follow. Each test case describes one game situation and you are to make a guess. On the first line of each test case, there are three integer numbers, P , C and M . P ($1 \leq P \leq 10$) is the number of pins, C ($1 \leq C \leq 100$) is the number of colours, and M ($1 \leq M \leq 100$) is the number of already played guesses.

Then there are $2 \times M$ lines, two lines for every guess. At the first line of each guess, there are P integer numbers representing colours of the guess. Each colour is represented by a number G_i , $1 \leq G_i \leq C$. The second line contains two integer numbers, B and W , stating for the number of black and white points given by the corresponding hint.

Let's have a secret code S_1, S_2, \dots, S_P and the guess G_1, G_2, \dots, G_P . Then we can make a set H containing pairs of numbers (I, J) such that $S_I = G_J$, and that any number can appear at most once on the first position and at most once on the second position. That means for every two

different pairs from that set, (I_1, J_1) and (I_2, J_2) , we have $I_1 \diamond I_2$ and $J_1 \diamond J_2$. Then we denote $B(H)$ the number of pairs in the set, that meet the condition $I = J$, and $W(H)$ the number of pairs with $I \diamond J$.

We define an ordering of every two possible sets H_1 and H_2 . Let's say $H_1 \leq H_2$ if and only if one of the following holds:

- $B(H_1) < B(H_2)$, or
- $B(H_1) = B(H_2)$ and $W(H_1) \leq W(H_2)$

Then we can find a maximal set H_{max} according to this ordering. The numbers $B(H_{max})$ and $W(H_{max})$ are the black and white points for that hint.

Output Specification

For every test case, print the line containing P numbers representing P colours of the next guess. Your guess must be valid according to all previous guesses and hints. The guess is valid if the sequence could be a secret code, i.e. the sequence was not eliminated by previous guesses and hints.

If there is no valid guess possible, output the sentence `You are cheating!`. If there are more more valid guesses, output the one that is lexicographically smallest. I.e. find such guess G that for every other valid guess V there exists such a number I that:

- $G_J = V_J$ for every $J < I$, and
- $G_I < V_I$.

Sample Input

```

3
4 3 2
1 2 3 2
1 1
2 1 3 2
1 1
4 6 2
3 3 3 3
3 0
4 4 4 4
2 0
8 9 3
1 2 3 4 5 6 7 8
0 0
2 3 4 5 6 7 8 9
1 0
3 4 5 6 7 8 9 9
2 0

```

Sample Output

```
1 1 1 3
You are cheating!
9 9 9 9 9 9 9 9
```

Hotline

Every customer sometimes needs help with new and unusual products. Therefore, hotline service is very important for every company. We need a single phone number where the customer can always find a friendly voice ready to help with anything. On the other hand, many people are needed to serve as hotline operators, and human resources are always very expensive. Moreover, it is not easy to pretend "friendly voice" at 4am and explain to a drunken man that you are really unable to give him the number to House of Parliament. It was also found that some of the questions repeat very often and it is very annoying to answer them again and again.

ACM is a modern company, wanting to solve its hotline problem. They want to decrease the number of human operators by creating a complex software system that would be able to answer most common questions. The customer's voice is analysed by a special Voice Recognition Module (VRM) and converted to a plain text. The text is then taken by an Artificial Automatic Adaptive Answering Algorithm (AAAAA). The most common questions are recognised and answered automatically. The replies are then converted to a sound by Text-to-Speech Module (TTS).

You are to write the AAAAA module. Because your algorithm should be adaptive, it has no explicit knowledge base. But it must be able to listen to sentences in English and remember the mentioned facts. Whenever the question is asked about such a fact, the system has to answer it properly. The VRM and TTS modules are already implemented, so the input and output of AAAAA will be in the text form.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of dialogues to follow. Each dialogue consists of zero or more lines, each of them containing one sentence: either statement or question. The statement ends with a dot character (.), the question ends with a question mark (?). No statement will appear more than once, however the questions can be repeated. There is one extra line after each dialogue. That line ends with an exclamation mark (!).

Sentences can contain words, spaces and punctuation characters (such as commas, colons, semicolons etc.). All words contain only letters of English alphabet and are case-sensitive. That means the same word is always written the same way, usually in lowercase. Acronyms, names and some other words can begin with capital letters. For simplicity, all sentences begin with a lowercase letter. Only if the first word should be written with a capital, the sentence begins with a capital letter. There are no unneeded spaces between words. No line will have more than 100 characters. There will be at most 100 statements per each test case.

Statements

Each statement has one of the following two forms ($_$ denotes a space):

subject $_$ *predicate*[s] [$_$ *object*] .

subject $_$ don't|doesn't $_$ *predicate* [$_$ *object*] .

The square brackets mark an optional part, the vertical line two possible variants. Subject is a single word, noun or pronoun in singular. Predicate is a verb (single word) denoting some activity. Object can be any text. Object does not contain any dots. Any pair "verb + object" determines unique activity. The same verb with different objects makes different independent activities, i.e. the different and independent meaning of the sentence. Sentence without any object can be considered as sentence with an empty object. The verb without an object has different and independent meaning than the same verb with any non-empty object.

The first variant of sentence denotes a positive statement. The word "*predicate*[s]" means verb that matches the subject of the sentence. If the subject is "I" or "you", the verb has the same form as the infinitive. With any other subject, the letter "s" is appended on the end of the verb. Assume there are no irregular verbs.

The second variant is a negative statement. Verb "don't" or "doesn't" must also match the subject. The form "don't" is used with either "I" or "you", "doesn't" is used in any other case.

A special generic subject "everybody" can be used. It means the activity holds for any subject. Other special subject is "nobody". Such sentence also holds for any subject, but its meaning is negative. Both of these generic subjects can be used with the first variant only (without "doesn't"). The sentence "nobody likes something" is exactly equal to "everybody doesn't like something", except the latter form will never occur in the input.

Questions

Each question has one of the following three forms:

1. do|does $_$ *subject* $_$ *predicate* [$_$ *object*] ?
2. who $_$ *predicates* [$_$ *object*] ?
3. what $_$ do|does $_$ *subject* do ?

The word "do|does" always matches the subject ("do I?", "do you?", "does any other subject?"). In the second type of question, predicate always matches the word "who", i.e. the "s" is always appended. Generic subjects cannot be used in questions.

Output Specification

For each dialogue, your program must output the line Dialogue #*D*:, where *D* is the sequence number of dialogue, starting with 1. Then print exactly three lines for every question: the first line repeats the question, the second line contains the answer, and the third line is empty. Print nothing for statements. After each dialogue, print the same line with

an exclamation mark that was in the input. Then print one extra empty line. Empty line contains a new-line character only.

The answer must be properly formatted to be accepted by a TTS module. Only the statements appearing in the input before the answer are used for the corresponding reply. If there is any contradiction among statements, the reply is always `I am abroad.`. If the question and statements consider the special subject "you", it must be replaced with "I" in the answer. If the question considers special subject "I", it must be replaced with "you" in the answer. The verb must always match the subject of the sentence. The exact form of the correct answer depends on the type of question.

1. does subject predicate [object] ?

If there is any positive statement about the mentioned subject (or generic subject "everybody"), predicate and object, the answer is:
`yes , \neg subject \neg predicate[s] [\neg object] .`

If there is any negative statement about the mentioned subject (or generic subject "nobody"), predicate and object, the answer is:
`no , \neg subject \neg don't | doesn't \neg predicate [\neg object] .`

Otherwise, the answer is: `maybe .`

Subject in the answer is always the same subject as the subject of the question.

2. who predicates [object] ?

If there is a positive statement considering any subject, the specified predicate and object, the answer is:
`subject \neg predicate[s] [\neg object] .`

If two or more subjects match the activity, replace the subject in the answer with enumeration of all such subjects, in the same order as the corresponding statements have appeared in the input. Subjects are separated with comma and space, last two subjects are separated with the word "and". If "everybody" belongs to the group of enumerated subjects, do not enumerate subjects, and print "everybody" only. If the enumeration contains at least two subjects, the predicate matches the plural subject (i.e. verb is without trailing "s"), otherwise it matches the only subject.

`subject1 , \neg subject2 \neg and \neg subject3 predicate [\neg object] .`

If there is a negative statement considering the generic subject "nobody", the specified predicate and object, the answer is:
`nobody \neg predicates [\neg object] .`

Otherwise, the answer is: `I don't know .`

3. what does subject do ?

If there are one or more sentences (both positive and negative) considering the specified

subject (or a generic subject "everybody" or "nobody"), all verbs and objects from such sentences must be included in a reply in the same order as the corresponding sentences have appeared in the input. No verb-object pair can be included more than once (the eventual second appearance must be skipped). The verb-object pairs are separated by a comma followed by a space, the last verb is separated by a comma and the word "and". Please note the comma is printed here although there was no comma when separating the subjects in the previous type of answer (see above). The negative answers have the same form as the statements, that means the verb "don't" or "doesn't" is used:

```
subject [ ⊎ don't|doesn't ] ⊎ predicate1[s] [ ⊎ object1 ] ,
[ ⊎ don't|doesn't ] ⊎ predicate2[s] [ ⊎ object2 ] ,
⊎ and [ ⊎ don't|doesn't ] ⊎ predicate3[s] [ ⊎ object3 ] .
subject [ ⊎ don't|doesn't ] ⊎ predicate1[s] [ ⊎ object1 ] ,
⊎ and [ ⊎ don't|doesn't ] ⊎ predicate2[s] [ ⊎ object2 ] .
subject [ ⊎ don't|doesn't ] ⊎ predicate[s] [ ⊎ object ] .
```

Otherwise, the answer is: I don't know.

Sample Input

```
1
I like hotdogs.
nobody likes to work.
everybody smiles.
what do I do?
who smiles?
what do you do?
does Joe smile?
do I like to work?
everybody hurts sometimes.
who walks there?
Michal walks there.
who walks there?
what does Michal do?
do you understand?
nobody walks there.
do you understand now?
bye!
```

Sample Output

```
Dialogue #1:
what do I do?
you like hotdogs, don't like to work, and smile.

who smiles?
everybody smiles.

what do you do?
I don't like to work, and smile.

does Joe smile?
yes, Joe smiles.

do I like to work?
no, you don't like to work.
```

who walks there?
I don't know.

who walks there?
Michal walks there.

what does Michal do?
Michal doesn't like to work, smiles, hurts sometimes, and walks there.

do you understand?
maybe.

do you understand now?
I am abroad.

bye!

I-Keyboard

Most of you have probably tried to type an SMS message on the keypad of a cellular phone. It is sometimes very annoying to write longer messages, because one key must be usually pressed several times to produce a single letter. It is due to a low number of keys on the keypad. Typical phone has twelve keys only (and maybe some other control keys that are not used for typing). Moreover, only eight keys are used for typing 26 letters of an English alphabet. The standard assignment of letters on the keypad is shown in the left picture:

1	2 abc	3 def
4 ghi	5 jkl	6 mno
7 pqrs	8 tuv	9 wxyz
*	0 <i>space</i>	#

1	2 abcd	3 efg
4 hijk	5 lm	6 nopq
7 rs	8 tuv	9 wxyz
*	0 <i>space</i>	#

There are 3 or 4 letters assigned to each key. If you want the first letter of any group, you press that key once. If you want the second letter, you have to press the key twice. For other letters, the key must be pressed three or four times. The authors of the keyboard did not try to optimise the layout for minimal number of keystrokes. Instead, they preferred the even distribution of letters among the keys. Unfortunately, some letters are more frequent than others. Some of these frequent letters are placed on the third or even fourth place on the standard keyboard. For example, *s* is a very common letter in an English alphabet, and we need four keystrokes to type it. If the assignment of characters was like in the right picture, the keyboard would be much more comfortable for typing average English texts.

ACM have decided to put an optimised version of the keyboard on its new cellular phone.

Now they need a computer program that will find an optimal layout for the given letter frequency. We need to preserve alphabetical order of letters, because the user would be confused if the letters were mixed. But we can assign any number of letters to a single key.

Input Specification

There is a single positive integer T on the first line of input. It stands for the number of test cases to follow. Each test case begins with a line containing two integers K, L ($1 \leq K \leq L \leq 90$) separated by a single space. K is the number of keys, L is the number of letters to be mapped onto those keys. Then there are two lines. The first one contains exactly K characters each representing a name of one key. The second line contains exactly L characters representing names of letters of an alphabet. Keys and letters are represented by digits, letters (which are case-sensitive), or any punctuation characters (ASCII code between 33 and 126 inclusively). No two keys have the same character, no two letters are the same. However, the name of a letter can be used also as a name for a key.

After those two lines, there are exactly L lines each containing exactly one positive integer F_1, F_2, \dots, F_L . These numbers determine the frequency of every letter, starting with the first one and continuing with the others sequentially. The higher number means the more common letter. No frequency will be higher than 100000.

Output Specification

Find an optimal keyboard for each test case. Optimal keyboard is such that has the lowest "price" for typing average text. The *price* is determined as the sum of the prices of each letter. The price of a letter is a product of the letter frequency (F_i) and its position on the key. The order of letters cannot be changed, they must be grouped in the given order.

If there are more solutions with the same price, we will try to maximise the number of letters assigned to the last key, then to the one before the last one etc.

More formally, you are to find a sequence P_1, P_2, \dots, P_L representing the position of every letter on a particular key. The sequence must meet following conditions:

- $P_1 = 1$
- for each $i > 1$, either $P_i = P_{i-1} + 1$ or $P_i = 1$
- there are at most K numbers P_i such that $P_i = 1$
- the sum of products $S_P = \sum_{i=1}^L F_i \cdot P_i$ is minimal
- for any other sequence Q meeting these criteria and with the same sum $S_Q = S_P$, there exists such $M, 1 \leq M \leq L$ that for any $J, M < J \leq L, P_J = Q_J$, and $P_M > Q_M$.

The output for every test case must start with a single line saying `Keypad #I:`, where I is a sequential order of the test case, starting with 1. Then there must be exactly K lines, each representing one letter, in the same order that was used in input. Each line must contain the character representing the key, a colon, one space and a list of letters assigned to that particular key. Letters are not separated from each other.

Print one blank line after each test case, including the last one.

Sample Input

```
1
8 26
23456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
3371
589
1575
1614
6212
971
773
1904
2989
123
209
1588
1513
2996
3269
1080
121
2726
3083
4368
1334
518
752
427
733
871
```

Sample Output

```
Keypad #1:
2: ABCD
3: EFG
4: HIJK
5: LM
6: NOPQ
7: RS
8: TUV
9: WXYZ
```

