# Problem A
## Modular multiplication of polynomials

Consider polynomials whose coefficients are 0 and 1. Addition of two polynomials is achieved by 'adding' the coefficients for the corresponding powers in the polynomials. The addition of coefficients is performed by addition modulo 2, i.e., (0 + 0) mod 2 = 0, (0 + 1) mod 2 = 1, (1 + 0) mod 2 = 1, and (1 + 1) mod 2 = 0. Hence, it is the same as the exclusive-or operation.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

Subtraction of two polynomials is done similarly. Since subtraction of coefficients is performed by subtraction modulo 2 which is also the exclusive-or operation, subtraction of polynomials is identical to addition of polynomials.

$$(x^6 + x^4 + x^2 + x + 1) - (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

Multiplication of two polynomials is done in the usual way (of course, addition of coefficients is performed by addition modulo 2).

$$(x^6 + x^4 + x^2 + x + 1)\,(x^7 + x + 1)$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

Multiplication of two polynomials f(x) and g(x) modulo a polynomial h(x) is the remainder of f(x)g(x) divided by h(x).

$$(x^6 + x^4 + x^2 + x + 1)\,(x^7 + x + 1) \text{ modulo } (x^8 + x^4 + x^3 + x + 1)$$
$$= x^7 + x^6 + 1$$

The largest exponent of a polynomial is called its degree. For example, the degree of $x^7 + x^6 + 1$ is 7.

Given three polynomials f(x), g(x), and h(x), you are to write a program that computes f(x)g(x) modulo h(x). We assume that the degrees of both f(x) and g(x) are less than the degree of h(x). The degree of a polynomial is less than 1000.

Since coefficients of a polynomial are 0 or 1, a polynomial can be represented by d+1 and a bit string of length d+1, where d is the degree of the polynomial and the bit string represents the coefficients of the polynomial. For example, $x^7 + x^6 + 1$ can be represented by

```
8 1 1 0 0 0 0 0 1.
```

## Input

The input consists of T test cases. The number of test cases (T) is given in the first line of the input file. Each test case consists of three lines that contain three polynomials f(x), g(x), and h(x), one per line. Each polynomial is represented as described above.

## Output

The output should contain the polynomial f(x)g(x) modulo h(x), one per line.

| Sample Input | Output for the Sample Input |
|---|---|
| 2<br>7 1 0 1 0 1 1 1<br>8 1 0 0 0 0 0 1 1<br>9 1 0 0 0 1 1 0 1 1<br>10 1 1 0 1 0 0 1 0 0 1<br>12 1 1 0 1 0 0 1 1 0 0 1 0<br>15 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 | 8 1 1 0 0 0 0 0 1<br>14 1 1 0 1 1 0 0 1 1 1 0 1 0 0 |

# Problem B – Entropy

## Background

An entropy encoder is a data encoding method that achieves lossless data compression by encoding a message with "wasted" or "extra" information removed. In other words, entropy encoding removes information that was not necessary in the first place to accurately encode the message. A high degree of entropy implies a message with a great deal of wasted information; english text encoded in ASCII is an example of a message type that has very high entropy. Already compressed messages, such as JPEG graphics or ZIP archives, have very little entropy and do not benefit from further attempts at entropy encoding.

English text encoded in ASCII has a high degree of entropy because all characters are encoded using the same number of bits, eight. It is a known fact that the letters E, L, N, R, S and T occur at a considerably higher frequency than do most other letters in english text. If a way could be found to encode just these letters with four bits, then the new encoding would be smaller, would contain all the original information, and would have less entropy. ASCII uses a fixed number of bits for a reason, however: it's easy, since one is always dealing with a fixed number of bits to represent each possible glyph or character. How would an encoding scheme that used four bits for the above letters be able to distinguish between the four-bit codes and eight-bit codes? This seemingly difficult problem is solved using what is known as a "prefix-free variable-length" encoding.

In such an encoding, any number of bits can be used to represent any glyph, and glyphs not present in the message are simply not encoded. However, in order to be able to recover the information, no bit pattern that encodes a glyph is allowed to be the prefix of any other encoding bit pattern. This allows the encoded bitstream to be read bit by bit, and whenever a set of bits is encountered that represents a glyph, that glyph can be decoded. If the prefix-free constraint was not enforced, then such a decoding would be impossible.

Consider the text "AAAAABCD". Using ASCII, encoding this would require 64 bits. If, instead, we encode "A" with the bit pattern "00", "B" with "01", "C" with "10", and "D" with "11" then we can encode this text in only 16 bits; the resulting bit pattern would be "0000000000011011". This is still a fixed-length encoding, however; we're using two bits per glyph instead of eight. Since the glyph "A" occurs with greater frequency, could we do better by encoding it with fewer bits? In fact we can, but in order to maintain a prefix-free encoding, some of the other bit patterns will become longer than two bits. An optimal encoding is to encode "A" with "0", "B" with "10", "C" with "110", and "D" with "111". (This is clearly not the *only* optimal encoding, as it is obvious that the encodings for B, C and D could be interchanged freely for any given encoding without increasing the size of the final encoded message.) Using this encoding, the message encodes in only 13 bits to "0000010110111", a compression ratio of 4.9 to 1 (that is, each bit in the final encoded message represents as much information as did 4.9 bits in the original encoding). Read through this bit pattern from left to right and you'll see that the prefix-free encoding makes it simple to decode this into the original text even though the codes have varying bit lengths.

As a second example, consider the text "THE CAT IN THE HAT". In this text, the letter "T" and the space character both occur with the highest frequency, so they will clearly have the shortest encoding bit patterns in an optimal encoding. The letters "C", "I' and "N" only occur once, however, so they will have the longest codes. There are many possible sets of prefix-free variable-length bit patterns that would yield the optimal encoding, that is, that would allow the text to be encoded in the fewest number of bits. One such optimal encoding is to encode spaces with "00", "A" with "100", "C" with "1110", "E" with "1111", "H" with "110", "I" with "1010", "N" with "1011" and "T" with "01". The optimal encoding therefore requires only 51 bits compared to the 144 that would be necessary to encode the message with 8-bit ASCII encoding, a compression ratio of 2.8 to 1.

## Input

The input file will contain a list of text strings, one per line. The text strings will consist only of uppercase alphanumeric characters and underscores (which are used in place of spaces). The end of the input will be signalled by a line containing only the word "END" as the text string. This line should not be processed.

## Output

For each text string in the input, output the length in bits of the 8-bit ASCII encoding, the length in bits of an optimal prefix-free variable-length encoding, and the compression ratio accurate to one decimal point.

## Example

| Input | Output |
| --- | --- |
| AAAAABCD<br>THE_CAT_IN_THE_HAT<br>END | 64 13 4.9<br>144 51 2.8 |

## Problem G: Gone Fishing

John is going on a fishing trip. He has $h$ hours available ($1 \leq h \leq 16$), and there are $n$ lakes in the area ($2 \leq n \leq 25$) all reachable along a single, one-way road. John starts at lake 1, but he can finish at any lake he wants. He can only travel from one lake to the next one, but he does not have to stop at any lake unless he wishes to. For each $i = 1, \ldots, n-1$, the number of 5-minute intervals it takes to travel from lake $i$ to lake $i+1$ is denoted $t_i$ ($0 < t_i \leq 192$). For example, $t_3 = 4$ means that it takes 20 minutes to travel from lake 3 to lake 4.

To help plan his fishing trip, John has gathered some information about the lakes. For each lake $i$, the number of fish expected to be caught in the initial 5 minutes, denoted $f_i$ ($f_i \geq 0$), is known. Each 5 minutes of fishing decreases the number of fish expected to be caught in the next 5-minute interval by a constant rate of $d_i$ ($d_i \geq 0$). If the number of fish expected to be caught in an interval is less than or equal to $d_i$, there will be no more fish left in the lake in the next interval. To simplify the planning, John assumes that no one else will be fishing at the lakes to affect the number of fish he expects to catch.

Write a program to help John plan his fishing trip to maximize the number of fish expected to be caught. The number of minutes spent at each lake must be a multiple of 5.

### Input

You will be given a number of cases in the input. Each case starts with a line containing $n$. This is followed by a line containing $h$. Next, there is a line of $n$ integers specifying $f_i$ ($1 \leq i \leq n$), then a line of $n$ integers $d_i$ ($1 \leq i \leq n$), and finally, a line of $n-1$ integers $t_i$ ($1 \leq i \leq n-1$). Input is terminated by a case in which $n = 0$.

### Output

For each test case, print the number of minutes spent at each lake, separated by commas, for the plan achieving the maximum number of fish expected to be caught (you should print the entire plan on one line even if it exceeds 80 characters). This is followed by a line containing the number of fish expected. If multiple plans exist, choose the one that spends as long as possible at lake 1, even if no fish are expected to be caught in some intervals. If there is still a tie, choose the one that spends as long as possible at lake 2, and so on. Insert a blank line between cases.

### Sample Input

```
2
1
10 1
2 5
2
4
4
10 15 20 17
0 3 4 3
1 2 3
4
4
10 15 50 30
0 3 4 3
1 2 3
0
```

### Sample Output

```
45, 5
Number of fish expected: 31

240, 0, 0, 0
Number of fish expected: 480

115, 10, 50, 35
Number of fish expected: 724
```

# Problem D

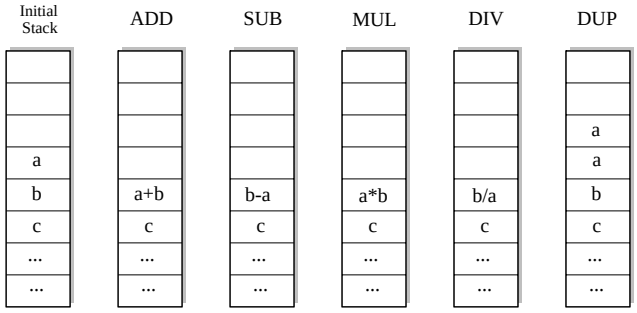
## Optimal Programs

As you know, writing programs is often far from being easy. Things become even harder if your programs have to be as fast as possible. And sometimes there is reason for them to be. Many large programs such as operating systems or databases have "bottlenecks" – segments of code that get executed over and over again, and make up for a large portion of the total running time. Here it usually pays to rewrite that code portion in assembly language, since even small gains in running time will matter a lot if the code is executed billions of times.

In this problem we will consider the task of automating the generation of optimal assembly code. Given a function (as a series of input/output pairs), you are to come up with the shortest assembly program that computes this function.

The programs you produce will have to run on a stack based machine, that supports only five commands: `ADD`, `SUB`, `MUL`, `DIV` and `DUP`. The first four commands pop the two top elements from the stack and push their sum, difference, product or integer quotient[1], respectively, on the stack. The `DUP` command pushes an additional copy of the top-most stack element on the stack.

So if the commands are applied to a stack with the two top elements *a* and *b* (shown to the left), the resulting stacks look as follows:




At the beginning of the execution of a program, the stack will contain a single integer only: the input. At the end of the computation, the stack must also contain only one integer; this number is the result of the computation.

There are three cases in which the stack machine enters an error state:

- A `DIV`-command is executed, and the top-most element of the stack is 0.
- A `ADD`, `SUB`, `MUL` or `DIV`-command is executed when the stack contains only one element.
- An operation produces a value greater than 30000 in absolute value.

[1] This corresponds to `/` applied to two integers in C/C++, and `DIV` in Pascal.

## Input

The input consists of a series of function descriptions. Each description starts with a line containing a single integer $n$ ($n \leq 10$), the number of input/output pairs to follow. The following two lines contains $n$ integers each: $x_1, x_2, \ldots, x_n$ in the first line (all different), and $y_1, y_2, \ldots, y_n$ in the second line. The numbers will be no more than 30000 in absolute value.

The input is terminated by a test case starting with $n = 0$. This test case should not be processed.

## Output

You are to find the shortest program that computes a function $f$, such that $f(x_i) = y_i$ for all $i \in \{1, \ldots, n\}$. This implies that the program you output may not enter an error state if executed on the inputs $x_i$ (although it may enter an error state for other inputs). Consider only programs that have at most 10 statements.

For each function description, output first the number of the description. Then print out the sequence of commands that make up the shortest program to compute the given function. If there is more than one such program, print the lexicographically smallest. If there is no program of at most 10 statements that computes the function, print the string "`Impossible`". If the shortest program consists of zero commands, print "`Empty Sequence`".

Output a blank line after each test case.

## Sample Input

```
4
1 2 3 4
0 -2 -6 -12
3
1 2 3
1 11 1998
1
1998
1998
0
```

## Sample Output

```
Program 1
DUP DUP MUL SUB

Program 2
Impossible

Program 3
Empty sequence
```

## Problem F: Roads Scholar

The Hines Sign company has been contracted to supply roadside signs for the state highway system. The head of the company has put his son Myles Hines in charge of one particular class of signs, those which indicate the number of miles to various towns. Myles is given a layout of the highway system and a set of locations where the signs should go; he is in charge of determining the mileage to nearby cities. To select which cities should be listed on any sign, he uses the following strategy: City X is put on the sign if the sign is on a road such that the shortest path from the intersection immediately preceeding the sign to X uses the road. You may assume that there is only one shortest path between each pair of intersections.

### Input

Input consists of a single problem instance consisting of a description of the highway system, followed by a set of sign locations. The highway system is defined as a set of intersections (some of which are also city locations) and a set of roads connecting the intersections. The first line of a problem instance contains three positive integers $n$ $m$ $k$: $n$ specifies the number of intersections (numbered $0, 1, 2, \cdots, n-1$), $m$ indicates the number of roads between connections, and $k$ indicates the number of intersections which are also cities. Following this are $m$ lines of the form $i_1$ $i_2$ $d$, which specifies a two-way road between intersections $i_1$ and $i_2$ of distance $d$. The next $k$ lines are of the form $i$ *name*, which specifies that intersection $i$ is a city called *name*. After this is a line with a single positive integer $s$ indicating the number of signs to place on the highway. The remaining $s$ lines are of the form $i_1$ $i_2$ $d$, indicating that a sign is to be placed on the road going from $i_1$ to $i_2$ a distance $d$ from $i_1$ ($d$ will always be non-zero and less than the distance from $i_1$ to $i_2$). For all problem instances, the length of *name* will be $\leq 18$ characters, and $5 \leq n \leq 30$. All distances will be non-zero and to the nearest hundredth mile.

### Output

Each sign should be output as follows:

$name_1$ $d_1$
$name_2$ $d_2$
...

where each $name_i$ should be left justified in a field of width 20, and each distance $d_i$ is rounded to the nearest mile. (Round .50 up. For example, 7.50 should be rounded to 8.) Each name-distance pair should be sorted by the rounded distance; pairs with the same rounded distance should be printed in alphabetical order. Signs should be output in the order in which they were listed in the input, and you should separate each sign output with a blank line. You may assume that every sign will have at least one city listed on it.

### Sample Input

```
8 17 4
0 1 7.12
0 2 8.34
0 3 5.33
0 4 5.36
1 2 4.21
1 6 6.99
1 7 10.26
2 3 2.74
2 6 5.04
3 4 4.12
3 5 7.72
3 6 5.71
4 5 8.94
4 6 10.29
5 6 5.47
5 7 8.55
6 7 6.01
0 Allentown
1 Bobtown
6 Charlestown
7 Downville
3
0 3 2.17
3 2 0.45
4 3 3.14
```

### Sample Output

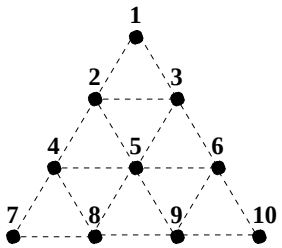
```
Charlestown         9
Downville           15

Bobtown             7

Charlestown         7
Bobtown             8
Downville           13
```

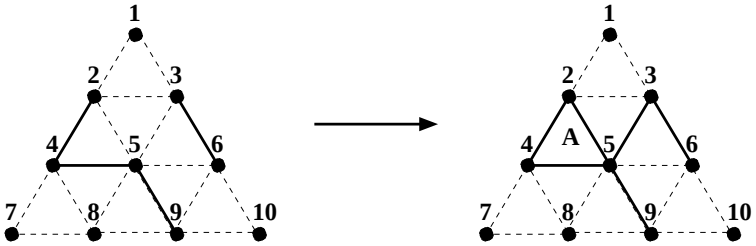
## Problem A: Triangle War

Triangle War is a two-player game played on the following triangular grid:




Two players, A and B, take turns filling in any dotted line connecting two dots, with A starting first. Once a line is filled, it cannot be filled again. If the line filled by a player completes one or more triangles, she owns the completed triangles and she is awarded another turn (i.e. the opponent skips a turn). The game ends after all dotted lines are filled in, and the player with the most triangles wins the game. The difference in the number of triangles owned by the two players is not important.

For example, if A fills in the line between 2 and 5 in the partial game on the left below:




Then, she owns the triangle labelled A and takes another turn to fill in the line between 3 and 5. B can now own 3 triangles (if he wishes) by filling in the line between 2 and 3, then the one between 5 and 6, and finally the one between 6 and 9. B would then make one more move before it is A's turn again.

In this problem, you are given a number of moves that have already been made. From the partial game, you should determine which player will win assuming that each player plays a perfect game from that point on. That is, assume that each player always chooses the play that leads to the best possible outcome for himself/herself.

### Input

You will be given a number of games in the input. The first line of input is a positive integer indicating the number of games to follow. Each game starts with an integer $6 \leq m \leq 18$ indicating the number of moves that have been made in the game. The next $m$ lines indicate the moves made by the two players in order, each of the form $i \quad j$ (with $i < j$) indicating that the line between $i$ and $j$ is filled in that move. You may assume that all given moves are legal.

### Output

For each game, print the game number and the result on one line as shown below. If A wins, print the sentence "A wins." If B wins, print "B wins."

### Sample Input

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

### Sample Output

```
Game 1: B wins.
Game 2: A wins.
Game 3: A wins.
Game 4: B wins.
```