Algorithmique effective – DM 8
Pour le mardi 18 avril 2006

1 Transportation (UVa: 301)

Ruratania is just entering capitalism and is establishing new enterprising activities in many fields including transport. The transportation company TransRuratania is starting a new express train from city A to city B with several stops in the stations on the way. The stations are successively numbered, city A station has number 0, city B station number m. The company runs an experiment in order to improve passenger transportation capacity and thus to increase its earnings. The train has a maximum capacity n passengers. The price of the train ticket is equal to the number of stops (stations) between the starting station and the destination station (including the destination station). Before the train starts its route from the city A, ticket orders are collected from all onroute stations. The ticket order from the station S means all reservations of tickets from S to a fixed destination station. In case the company cannot accept all orders because of the passenger capacity limitations, its rejection policy is that it either completely accept or completely reject single orders from single stations.

Write a program which for the given list of orders from single stations on the way from A to B determines the biggest possible total earning of the TransRuratania company. The earning from one accepted order is the product of the number of passengers included in the order and the price of their train tickets. The total earning is the sum of the earnings from all accepted orders.

Input

The input file is divided into blocks. The first line in each block contains three integers: passenger capacity n of the train, the number of the city B station and the number of ticket orders from all stations. The next lines contain the ticket orders. Each ticket order consists of three integers: starting station, destination station, number of passengers. In one block there can be maximum 22 orders. The number of the city B station will be at most 7. The block where all three numbers in the first line are equal to zero denotes the end of the input file.

Output

The output file consists of lines corresponding to the blocks of the input file except the terminating block. Each such line contains the biggest possible total earning.

Sample Input

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

Sample Output

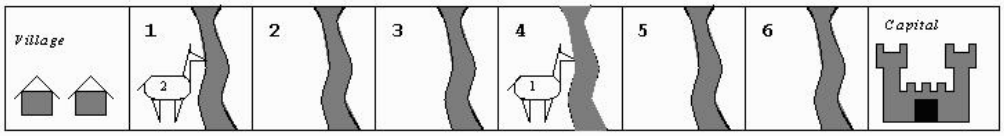
```
19
34
```

2 Donkey (UVa: 888)

Long ago, a number of farmers wanted to sell their donkeys. These farmers lived in a small village and the marketplace was in the capital. There was only one small road from the village the farmers lived in to the capital, and one day the farmers left for the long journey towards the capital. Every farmer wanted to travel as fast as possible to be the first to sell his donkey, but there was one major problem: the donkeys were very stubborn; everytime a donkey reached a river there was a chance that he would stay there for some time, not willing to cross the river. However, two donkeys never rested at the same river.

The game 'Donkey' is derived from this legend: N players (numbered from $1 \ldots N$) start with their donkey in the village. Between the village and the capital are M rivers. The players take turns in throwing a fair, 6-sided die and move their donkey the thrown number of rivers towards the capital. Since only one donkey can rest at a river, the donkey is placed at the next free river if necessary. After player 1 has thrown the die, it is player 2's turn, etc. After N comes 1. The player that passes all rivers first wins the game.

This is an example of a 'Donkey' gameboard, where $N = 2$ and $M = 6$. Player 1's donkey is located at river 4 and player 2's donkey is located at river 1.



Your task is to give the probability that player 1 wins the game, given a certain board position.

Input

The first line contains a single integer which equals the number of test cases that follow. Each of the following lines contains one test case. The first integer on a line gives the number of rivers M, the second integer gives the number of players N ($1 \leq N \leq 4$ and $NM \leq 50$). Then follow N integers P_i ($0 \leq P_i \leq M$, $1 \leq i \leq N$), representing the position of the i-th player. The river closest to the village has number 1, the river closest to the capital has number M. The village has number 0. Two donkeys may not be positioned at the same river. However, more than one donkey may be standing in the village. The last integer on the line gives the player, whose turn it is.

Output

For every situation you have to print the following line, giving the game number (1 for the first, 2 for the second, ...) and the probability player 1 wins with an accuracy of 3 decimals:

```
Game N:the probability that player 1 wins = D.DDD
```

Sample Input

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

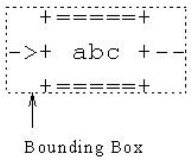
Sample Output

```
Game 1:the probability that player 1 wins = 0.667
Game 2:the probability that player 1 wins = 0.093
Game 3:the probability that player 1 wins = 0.366
```

3 Syntrax (UVa: 891)

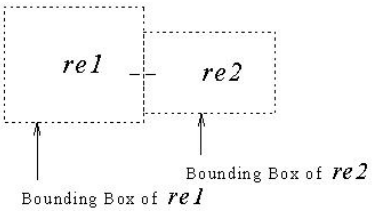
Given a simple syntax for describing regular expressions, one can find a graphical representation for a given regular expression using ASCII characters like '`-`', '`+`', and '`/`'. The syntax we use can be drawn by using four different patterns:

1. `"abc"` is the terminal string `'abc'`, represented as



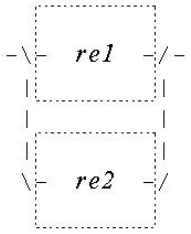
   Note that the graphical representation of every expression has a bounding box. This is the smallest rectangle that surrounds the graphic.

2. `(re1 re2)` is a sequence of first expression `re1`, then expression `re2`, represented as



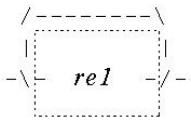
   The two expressions `re1` and `re2` have to be concatenated such that the bounding boxes of the two expressions touch and such that the `'-'` on the right of `re1` matches the `'-'` on the left of `re2`.

3. {`re1 re2`} represents alternatives, either `re1` or `re2`, represented as



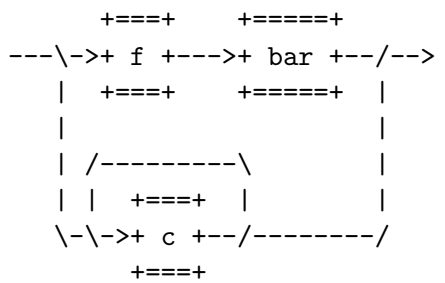
   the number of `'|'` characters that has to be added depends on the shapes of `re1` and `re2`. There has to be exactly one straight blank line between the bounding box of the graphical representation of `re1` and the bounding box of `re2`. If necessary, also a number of `'-'` characters has to be added on the right side of `re1` or `re2`, to make the drawing possible. Note that the `'-'` on the left of `re1` and `re2` matches the `'\'` character and that the `'-'` on the right of `re1` and `re2` matches the `'/'` character in the drawing.

4. `[re1]` is a 1-or-more repetition of `re1`, represented as



   Note that the `'-'` on the left of `re1` matches the `'\'` character and that the `'-'` on the right of `re1` matches the `'/'` character in the drawing.

For example the graphical representation for the regular expression {`("f" "bar") ["c"]`} looks like:



Write a program that reads syntax rules and prints the size of the graphical representation. For esthetic reasons, the entire graphic has a '-' on the left and a '->' on the right.

Input

The input consists of a line holding the number of test cases, followed by the input expressions (one per line). The expressions are formatted according to the following grammar:

```
expression :: sequence | alternatives | repetition | terminal
sequence :: ( ws expression expression ) ws
alternatives :: { ws expression expression } ws
repetition :: [ ws expression ] ws
terminal :: " character* " ws
ws :: (<space> | <tab>)*
character :: <any character except " and control-characters (ASCII 0..31)>
```

Note that the grammar is specified according to the following notational conventions:

```
x y sequence: x followed by y
x| y choice: x or y
x* repetition: zero or more occurrences of x
< > used for describing a character
```

Output

For each expression, output a line of the form `XxY` with `X` and `Y` the width and height of the graphical representation of that expression.

Sample Input

```
1
{("f" "bar") ["c"]}
```

Sample Output

```
28x8
```

4 Lead or Gold (UVa: 802)

How to make gold from lead has baffled alchemists for centuries. At the last Alchemists Club Meeting (ACM), a sensational breakthrough was announced. By mixing the three chemicals Algolene, Basicine and Cobolase in the correct ratio, one can create a mixture that transforms lead into gold. Since Algolene, Basicine and Cobolase (or A, B, C for short) are generally not sold individually, but rather mixed into solutions, this may not be easy as it seems.

Consider the following example. Two mixtures of Algolene, Basicine and Cobolase are available, in weights 1:2:3 and 3:7:1, respectively. By mixing these solutions in a ratio of 1:2 we obtain a solution of A, B, C with weights 7:16:5. But there is no way to combine these mixtures into a new one with weights 3:4:5. If we additionally had a solution of weights 2:1:2, then a 3:4:5 mixture would be possible by combining eight parts of 1:2:3, one part of 3:7:1 and five parts of 2:1:2. As you see in both examples, weights for each component must be integers.

Determining which mixing weights we can obtain from a given set of solutions is no trivial task. But, as the ACM has shown, it is possibly a very profitable one. You must write a program to find mixing ratios.

Input

The input file contains several test cases. The first line of each test case contains an integer n ($0 \leq n < 100$) that represents the number of mixtures in the test case. The next n lines each contain three non-negative integers a_i, b_i, c_i, specifying the weights $a_i : b_i : c_i$ of A, B, C in the i-th mixture. At least one of these integers is not 0 for each mixture, and none of them will be greater than 15000.

Finally, there is one line containing three non-negative integers a, b, c, which specify the weights $a : b : c$ in the desired solution. At least one of these integers is not 0.

The input file is terminated with the integer '0' on a line by itself following the last test case.

Output

For each test case, output the word `Mixture`, followed by the ordinal number of the test case. On the next line, if it is possible to obtain the desired solution by mixing the input solutions, output the word `Possible`. Otherwise, output the word `Impossible`. Print a blank line between consecutive test cases.

Sample Input

```
2
1 2 3
3 7 1
3 4 5
3
1 2 3
3 7 1
2 1 2
3 4 5
0
```

Sample Output

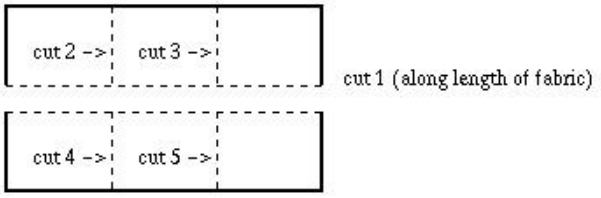
```
Mixture 1
Impossible

Mixture 2
Possible
```

5 Cutting Up, facultatif (UVa: 366)

Quilters often have to cut fabric into squares. To do this, they have a special tool (called a rotary cutter) that can cut through many layers of fabric at a time. The exact number of layers depends on the type of fabric being cut. In using a rotary cutter, the problem is not how long the cut needs to be (it is just as easy to make a cut one centimeter long as it is to make a cut 20 centimeters long), but how many cuts need to be made.

For example, to cut a 2 by 3 piece of fabric into 1 by 1 squares takes five cuts if only one layer of fabric may be cut at a time:



If two layers may be cut at a time, it will only take three cuts (the two pieces of fabric can be placed on top of each other after the first cut, so when the second and third cuts are made, they will also make the fourth and fifth cuts).

In cutting fabric, pieces may be placed on top of each other. Pieces do not need to be of identical shapes to be put on top of each other. For example, if the fabric above had first been cut vertically, a 1 by 2 piece could have been put on top of the 2 by 2 piece. Fabric may be rearranged between cuts. Fabric can not be folded. For example, to cut a 1 by 3 piece of fabric into squares will take 2 cuts, not 1 (if the fabric were folded in half before cutting). Quilters are thrifty people so they never have any waste: if a piece of fabric is m by n, they will get mn pieces from it.

Write a program to determine the fewest cuts necessary to cut a piece of fabric into squares.

Input

The input will consist of a number of lines. Each line will have 3 integers describing the fabric to be cut. The first is the maximum number of layers that can be cut at a time (between 1 and 200) and the last two are the dimensions (between 1 and 20 in each dimension).

A line consisting of the values 0 0 0 will mark the end of input. This line should not be processed.

Output

For each line, echo the dimensions and give the smallest number of cuts that are necessary to cut it entirely into 1 by 1 squares. Use the format given in the Sample Output below.

Leave a blank line after each line of output.

Sample Input

```
1 3 2
1 5 5
2 3 2
3 2 3
10 5 5
1 2 1
0 0 0
```

Sample Output

```
3 by 2 takes 5 cuts

5 by 5 takes 24 cuts

3 by 2 takes 3 cuts

2 by 3 takes 3 cuts

5 by 5 takes 6 cuts

2 by 1 takes 1 cuts
```