Problem A

Hardware

source: hardware.*

Ola Clason's Hardware store is an old company where most work is done "the old way". Among other things, the company is the one and only provider of marble house numbers. These house numbers have become extremely popular among construction companies, especially the ones building luxury estates. This is of course great for Ola Clason, but also a small problem. Nisse, who has been managing the incoming orders has turned out to be a bottleneck in Ola's business. Most orders are on the form "Coconut Drive 200, 202, 204, ..., 220". This means every even number between 200 and 220. Nisse's work is to transfer an order to a list of necessary digits and other symbols.

Your assignment is to write a program that automates Nisse's work with orders containing only positive integer house numbers. Nisse will still in the future process all special orders (those including non digit symbols) by hand.

Input specifications

On the first line of input is a single positive integer n, specifying the number of orders that follow. The first line of each order contains the road name for that order. No road name is longer than 50 characters. The second line states the total number of buildings needing new marble numbers on that order. Then follows the different house number specifications on several lines. These lines are of two kinds: *single number lines* and *multiple number lines*. A *single number line* simply consists of the house number by itself, while a *multiple number line* starts with a "+"-sign, followed by three positive integer numbers: first number, last number and the interval between the house numbers. The distance between the first and last house number will always be a multiple of the house number interval. A house number will never have more than five digits. After the last house number specification line, the next order follows, if there is any.

Output specifications

For each order, the output consists of 13 lines. The first and second lines should be identical with the first two input lines. Then, there follows 10 lines with information on how many marble digits of each kind the order consists of. These rows are on the format "Make X digit Y" where X is how many copies of digit Y they need to make. The last row states the total number Z of digits needed, on the format "In total Z digits". If there is only one digit to produce, it should say "In total 1 digit", in order to be grammatically correct.

Sample input

```
1
Short Street
23 addresses
+ 101 125 2
275
+ 100 900 100
```

Output for sample input

```
Short Street
23 addresses
Make 23 digit 0
Make 22 digit 1
Make 5 digit 2
Make 4 digit 3
Make 1 digit 4
Make 5 digit 5
Make 1 digit 6
Make 4 digit 7
Make 1 digit 8
Make 3 digit 9
In total 69 digits
```

Problem B

Budget

source: budget.*

I have a problem. I was supposed to make a budget proposal for this competition and I havn't started yet. It must be done at the end of the competition or the others will notice that I havn't done it. This is where I need you.

The budget proposal is a matrix where the rows represent the different kinds of expenses and the columns reptresent the different sites.

We had a meeting about this some time ago where we discussed the sums over different kind of expenses and sums over different sites. There was also some talk about special constraints: someone mentioned that Stockholm would need at least SEK 2000 for food and someone from Lund argued they wouldn't use more than SEK 100 for paperclips. Anyway, I'm sure there was more, I'll go and try to find some notes from that meeting if you write a program that can solve the problem when we get the data.

And, by the way, no one really reads budget proposals anyway, so you'll just have to make sure that it sums up properly and meets all constraints.

Input specifications

The first line of the input contains an integer N giving the number of test cases.

The first line of each test case contains two integers m, n giving the number of rows and columns ($m \leq 200, n \leq 20$). The second line contains m integers, giving the row sums of the matrix. The third line contains n integers, giving the column sums of the matrix. The fourth line contains an integer c giving the number of constraints. The next c lines contain the constraints.

Each constraint consists of two integers r, c specifying some entry (or entries) in the matrix (the upper left corner is 1 1 and 0 is interpreted as all, i.e. 4 0 means all entries on the fourth row and 0 0 means the entire matrix), one element from the set $\{<, =, >\}$ and one integer v, with the obvious interpretation. For instance, the constraint 1 2 $>$ 5 means that the entry in the 1st row and 2nd column must have an entry strictly greater than 5, and the constraint 4 0 = 3 means that all elements in the fourth row should be equal to 3.

Output specifications

For each case output a matrix of non-negative integers meeting the above constraints or the string "IMPOSSIBLE" if no legal solution exists.

Output an empty line between cases.

Sample input

```
2

2 3
8 10
5 6 7
4
0 2 > 2
2 1 = 3
2 3 > 2
2 3 < 5

2 2
4 5
6 7
1
1 1 > 10
```

Output for sample input

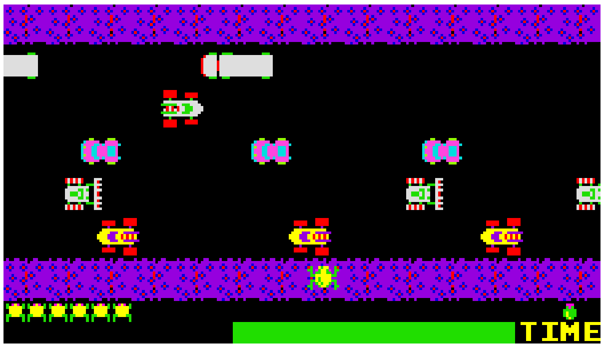
```
2 3 3
3 3 4

IMPOSSIBLE
```

Problem C

Frogger

source: frogger.*

"Frogger" was one of the first really popular arcade games after it was introduced by SEGA in 1981. The game consists of helping a frog cross a multi-lane motorway without getting run over by a car. You are given a view of an n-lane motorway where each lane consists of m different spaces that can either be empty or be occupied by a car. On each side of the motorway is a curb on which the frog can move freely. In the traffic lanes the frog can only move on the spaces not occupied by cars. The motorway is constructed in such a way that the direction in which the cars travel alternates between the lanes, with cars in the first lane (the one closest to the starting point of the frog) moving to the right. The cars never switch lanes and only move one step forward in each turn. To ensure a steady supply of traffic, a car that reaches the boundary of its lane is reentered at the opposite end of its lane.



In one turn of the game all the cars move one step in their assigned direction while the frog can either move one step to the right or to the left, or one step up or down (between lanes or between the curb and the adjoining lane), or it can stand still. Contrary to the cars the frog cannot "wrap-around" i.e. move in one step between the first and last position of a lane or a curb.

The frog and the cars move simultaneously. Thus the frog can move to a space given that there will be no car on it in the next round. If the frog is on the same space at the same time as a car it is run over and dies. Note that the frog can jump over an adjacent approaching car in the same lane as itself.

Your job is to write a computer program that will calculate the minimum number of turns needed for the frog to get from its starting position on the curb to its final position on the curb on the other side of the road or to determine that this is not possible within a given number of rounds.

Input specifications

First there will be a line containing the number of scenarios you are asked to help the frog in. For each scenario there will first be a line containing a positive integer $x \leq 10^5$ giving the maximum number of rounds that can be used. The next line contains the number of lanes n, $1 \leq n \leq 20$, and the length of each lane m, $1 \leq m \leq 50$. Each of the next $n + 2$ lines will contain a string of m characters. The character `X` indicates a car, the character `O` (letter O) indicates a free space, the character `F` gives the starting position of the frog, and the character `G` gives the final destination of the frog. The first line indicates the destination curb, consisting of `O`'s and exactly one `G` while the last line gives the starting curb consisting of `O`'s and exactly one `F`, while the intermediate lines each represent one lane of the motorway.

Output specifications

The output will be one line per scenario, either giving the minimum number of turns needed before the frog can get from its starting position to the final position without getting run over by a car or a statement indicating that this was not possible within the maximum number of allowed turns.

Sample input

```
2
10
4 4
OOGO
XXOO
XOOX
XXOO
XXOO
OOFO
2
2 2
OG
XX
OO
FO
```

Output for sample input

```
The minimum number of turns is 9.
The problem has no solution.
```

Problem D

Gallup

source: gallup.*

Often, we see results of gallups, like this:

```
Prefer red:     3.5%
Prefer green:   4.5%
Prefer yellow: 22.0%
Prefer blue:   70.0%
```

and you begin to wonder: how many people did they really ask? If the numbers are simple, like 20%, 40%, and 40%, you know that they asked 5 people (or 10, or 15, or more, but we are interested in the *minimum* number of people).

Your task is to write a program that reads sets of percentages and calculates the smallest number of people that could produce the given percentages. We know that this number is always less than 10 000.

Input specifications

The input is a set of percentages. Each set is on a line of its own. Every line starts with an integer n ($0 \leq n \leq 20$) giving the number of percentages in the set. If $n > 0$, the percentages follow as n numbers; these numbers may have 0–5 decimals, and all percentages in a set have the same number of decimals. (If there are no decimals, there is no decimal point.) The percentages always add up to about 100% as there may be small rounding errors. Numbers are rounded when digits are removed; they are rounded upwards if the first removed digit is 5 or more. Thus, 4.472 is rounded to 4.47, 4.5, or 4, depending on how many digits you want.

Output specifications

For each set of data, print a line starting with "`Case` *i*`:`", where "*i*" is the data set's number. Then follows a space and an integer giving the computed number of people. If no legal answer in the range 1–9999 exists, print "`error`" instead of the number.

Sample input

```
3 20 40 40
3 33.3 33.3 33.3
2 33 67
1 100.0000
4 3.75 4.25 22.00 70.00
2 49 51
2 50 51
2 49 50
0
```

Output for sample input

```
Case 1: 5
Case 2: 3
Case 3: 3
Case 4: 1
Case 5: 400
Case 6: 35
Case 7: 200
Case 8: error
```

Problem E

Jackpot

source: jackpot.*

Bill has found the perfect way to make money playing the slot machines. After months of careful research, he has finally figured out the mechanics behind how the machines operate. Now he is ready to make profit of his findings.

But first an introduction to the game. A slot machine consists of a number of wheels, usually three or four, each with a number of symbols printed on it – cherries, oranges, bells, etc. – and will show one of its symbols at a given time. To play, you insert a coin, push a button and the wheels start spinning. After spinning for a while, each wheel stops – at random it seems – at one of its symbols. If all wheels stop at the same symbol, or some nice combination of symbols, the player wins. One combination that is especially desirable is having the jackpot symbol on all wheels. This combination is simply called 'jackpot' and will make you rich for life.

What Bill has discovered is that each wheel will stop at the jackpot symbol with a certain periodicity, which differs a lot between wheels. He has also figured out (after some sneeking around at the slot-machine factory) that all newly manufactured slot-machines are delivered showing the jackpot combination, and that they all have a counter at the back, telling how many times the machine has been played. This counter is always set to zero at delivery.

Now, all Bill needs to do is to calculate the number of times a machine has to be played between two occurrences of the jackpot combination. We will call this number the jackpot periodicity. This is of course the same as the number of times the machine has to be played after leaving the factory, before it gives its first jackpot. Thus, with a glance at the counter on the back of a machine, Bill can figure out if it is about to give a jackpot.

As Bill knows that you are a skillful computer programmer, he turns to you with the problem of calculating the jackpot periodicity. For each machine, he will give you the number of wheels, and the periodicity with which the jackpot symbol shows up on each wheel.

Input specifications

One line with the number of machines $n \leq 20$. For each machine, one line with the number of wheels $w \leq 5$, and one line with w numbers, $p_1, ..., p_w$ the periodicity of each wheel $p_k \leq 1000$.

Output specifications

One line per machine: The jackpot periodicity of the machine, if it is less than or equal to a billion (10^9), otherwise output the text 'More than a billion.'.

Sample input

```
1
3
10 6 15
```

Output for sample input

```
30
```

Problem F

Spiderman's workout

source: spiderman.*

Staying fit is important for every super hero, and Spiderman is no exception. Every day he undertakes a climbing exercise in which he climbs a certain distance, rests for a minute, then climbs again, rests again, and so on. The exercise is described by a sequence of distances $d_1, d_2, \ldots, d_m$ telling how many meters he is to climb before the first first break, before the second break, and so on. From an exercise perspective it does not really matter if he climbs up or down at the i:th climbing stage, but it is practical to sometimes climb up and sometimes climb down so that he both starts and finishes at street level. Obviously, he can never be below street level. Also, he would like to use as low a building as possible (he does not like to admit it, but he is actually afraid of heights). The building must be at least 2 meters higher than the highest point his feet reach during the workout.

He wants your help in determining when he should go up and when he should go down. The answer must be legal: it must start and end at street level (0 meters above ground) and it may never go below street level. Among the legal solutions he wants one that minimizes the required building height. When looking for a solution, you may not reorder the distances.

If the distances are 20 20 20 20 he can either climb up, up, down, down or up, down, up, down. Both are legal, but the second one is better (in fact optimal) because it only requires a building of height 22, whereas the first one requires a building of height 42. If the distances are 3 2 5 3 1 2, an optimal legal solution is to go up, up, down, up, down, down. Note that for some distance sequences there is no legal solution at all (e.g., for 3 4 2 1 6 4 5).

Input specifications

The first line of the input contains an integer N giving the number of test scenarios. The following $2N$ lines specify the test scenarios, two lines per scenario: the first line gives a positive integer $M \leq 40$ which is the number of distances, and the following line contains the M positive integer distances. For any scenario, the total distance climbed (the sum of the distances in that scenario) is at most 1000.

Output specifications

For each input scenario a single line should be output. This line should either be the string "IMPOSSIBLE" if no legal solution exists, or it should be a string of length M containing only the characters "U" and "D", where the i:th character indicates if Spiderman should climb up or down at the i:th stage. If there are several different legal and optimal solutions, output one of them (it does not matter which one as long as it is optimal).

Sample input

```
3
4
20 20 20 20
6
3 2 5 3 1 2
7
3 4 2 1 6 4 5
```

Output for sample input

```
UDUD
UUDUDD
IMPOSSIBLE
```

Problem G

Subway planning

source: subway.*

The government in a foreign country is looking into the possibility of establishing a subway system in its capital. Because of practical reasons, they would like each subway line to start at the central station and then go in a straight line in some angle as far as necessary. You have been hired to investigate whether such an approach is feasible. Given the coordinates of important places in the city as well as the maximum distance these places can be from a subway station (possibly the central station, which is already built), your job is to calculate the minimum number of subway lines needed. You may assume that any number of subway stations can be built along a subway line.

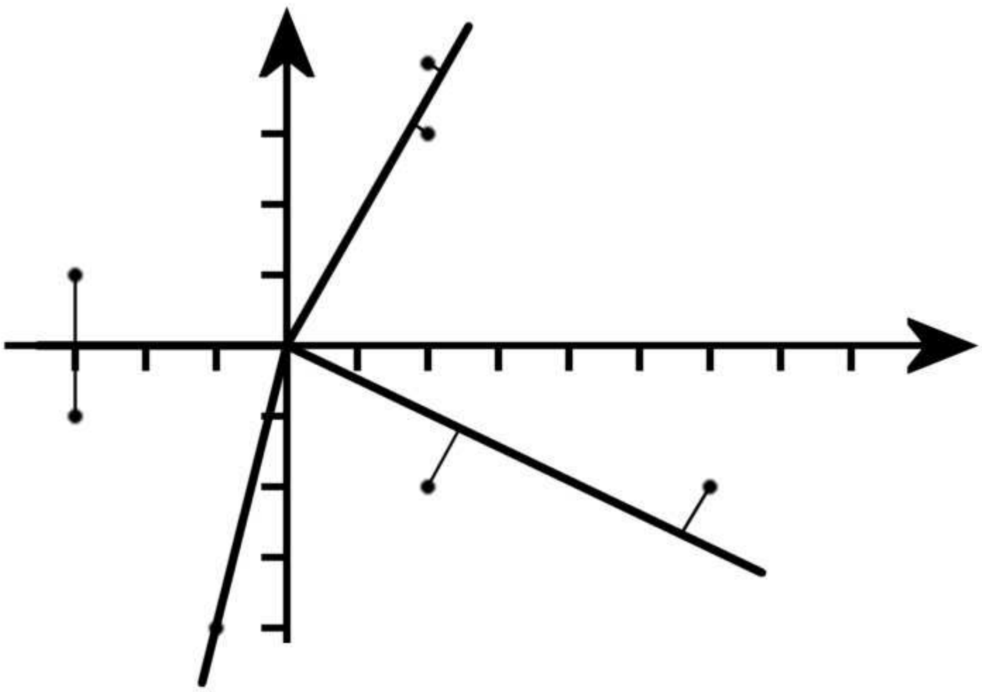


Figure 1: The figure above corresponds to the first data set in the example input.

Input specifications

The first line in the input file contains an integer N, the number of data sets to follow. Each set starts with two integers, n and d ($1 \leq n \leq 500$, $0 \leq d < 150$). n is the number of important places in the city that must have a subway station nearby, and d is the maximum distance allowed between an important place and a subway station. Then comes n lines, each line containing two integers x and y ($-100 \leq x, y \leq 100$), the coordinates of an important place in the capital. The central station will always have coordinates $0, 0$. All pairs of coordinates within a data set will be distinct (and none will be $0, 0$).

Output specifications

For each data set, output a single integer on a line by itself: the minimum number of subway lines needed to make sure all important places in the city is at a distance of at most d from a subway station.

Sample input

```
2
7 1
-1 -4
-3 1
-3 -1
2 3
2 4
2 -2
6 -2
4 0
0 4
-12 18
0 27
-34 51
```

Output for sample input

```
4
2
```

Problem H

Exploding CPU

source: `cpu.*`

The well known hardware manufacturing company *Processors for Professors* is about to release a highly specialized CPU with exceptional functionality in, amongst other areas, number theory. It has, for example, an instruction PFACT that takes one parameter and returns all prime factors of that parameter, with an outstanding execution speed. It has, however, one considerable problem. The scientists at the testing lab has just found out that the PFACT instruction for some special input values freaks out and makes the entire processor explode. Even though this could be an amusing effect, it is not the way it was intended to work. The skilled mathematicians have, by trial and error, found that the explosive numbers all share the same interesting number theoretic properties, which might be of help when troubleshooting.

An explosive number is a number $x = p_0 p_1 p_2 \ldots p_n$ where all p_is are distinct prime numbers such that $p_i = Ap_{i-1} + B$ for $i = 1, 2, \ldots, n$. $n \geq 3$, $p_0 \equiv 1$. A and B are always integers, and might be different for different explosive numbers.

For example, the processor will explode when factorizing the number 4505, because $4505 = 1 \cdot 5 \cdot 17 \cdot 53$ and $5 = 3 \cdot 1 + 2$, $17 = 3 \cdot 5 + 2$ and $53 = 3 \cdot 17 + 2$ and the numbers 5, 17 and 53 are all prime numbers. In this case $A = 3$ and $B = 2$.

You are kindly asked to write a computer program that will aid this company in estimating the impact of the errors, by calculating the amount of explosive numbers that exists within a given range of integers.

Input specifications

The input starts with a row containing the number $0 \leq N \leq 100$ of test cases that will follow. For each test case, there will be one row containing two integers, x_L and x_H separated by a single space. These numbers are such that $0 \leq x_L \leq x_H \leq 2,000,000,000$.

Output specifications

For each test case, output the number of explosive numbers that exist in the range $x_L \leq x \leq x_H$.

Sample input

```
2
4505 4505
0 5000
```

Output for sample input

```
1
5
```