Problem A
Vice City

source: `vicecity.c` *or* `vicecity.cpp` *or* `vicecity.java`

Description

"Tommy, there will be a programming contest here in Vice City. One of the coaches has stolen a copy of the problem set. The chief judge wants it back. Take out the coach guy at his hotel and return the problems back. The address is taped under the phone. Do it now!"

Not a tough job for you, Tommy Vercetti! Getting the mission at the pay phone, you must head off the coach at WK Charriot Hotel before he leaves. You have to get there fast! Get there very fast indeed! Unfortunately, the vehicle you start with may not run fast enough. But there are some fixed locations in Vice City at which you can find certain vehicles, like Diaz's Mansion where you can find an Infernus. This way, you may change your vehicle on your way to hotel several times. For example, in the first sample input, you ride from '`PayPhone`' to '`CarShowRoom`' on a '`PCJ600`' and drive the rest of the path in a '`HotRingRacer`'. Don't forget that it takes one minute each time you change your vehicle.

You are given the names of these locations in the city and the distances between each pair. At each location you can find a certain vehicle anytime you get there. Knowing the top speed of each vehicle, you want to find out the minimum time in which you can reach the hotel. For the sake of simplicity, assume that you always drive at top speed of your vehicle.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case has three parts. The first part consists of m lines ($1 \leq m \leq 100$) of the form '*vehicle speed*' where *vehicle* is the unique name of a vehicle and *speed* is a positive integer giving the top speed of the vehicle measured in Km/h.

The next part of the test case identifies the locations in the city and is separated from the first part by exactly one blank line. It consists of n lines ($2 \leq n \leq 500$) of the form '*location vehicle*' where *location* is the unique name of a location in the city and *vehicle* is the name of the vehicle available in that location. The list of locations always includes the starting location '`PayPhone`' and the destination '`WKCharriot`'.

The third part of the test case identifies the roads between locations and is separated from the previous part by exactly one blank line. It consists of several lines of the form '*loc*1 *loc*2 *distance*' indicating there is a (two-way) road of length *distance* between the locations *loc*1 and *loc*2. Distances are expressed in kilometers and are positive integers. The test case is terminated by a line containing a single asterisk character ('`*`').

All names (for vehicles and locations) are strings of at most 100 letters and digits with no space characters and are considered case sensitive. Items in an input line are separated by one or more space characters. Also, there may be arbitrary leading or trailing blanks except in empty lines used as separators.

Output

For each test case, there is one line in the output containing the minimum time (in minutes) you need to travel from '`PayPhone`' to '`WKCharriot`', or the word '`UNREACHABLE`' if the destination is unreachable from the starting point. Print the results as numbers with exactly three decimal digits after decimal point. That is, the possible decimal digits after the third one should be ignored, and if there are less than three digits after decimal point, zero digits should be printed for missing digits.

Sample

Input

```
2

Infernus 280
Cheetah 285
PCJ600 250
Stallion 180
HotRingRacer 300

Mansion Infernus
CarShowRoom HotRingRacer
VicePort Cheetah
NorthPointMall Infernus
PayPhone PCJ600
WKCharriot Stallion

PayPhone CarShowRoom 10
PayPhone VicePort 15
VicePort WKCharriot 20
CarShowRoom Mansion 15
Mansion WKCharriot 15
Mansion NorthPointMall 5
NorthPointMall WKCharriot 5
*
Caddy 80
MrWhoopie 60
Stretch 120
CubanHermes 160
Voodoo 170

CherryPoppy MrWhoopie
Mansion Stretch
PayPhone CubanHermes
LittleHaiti Voodoo
WKCharriot Caddy

PayPhone CherryPoppy 10
CherryPoppy LittleHaiti 15
Mansion WKCharriot 20
*
```

Output

```
8.400
UNREACHABLE
```

Problem B
House Numbers

source: `house.c` *or* `house.cpp` *or* `house.java`

Description

NarmakSung has a hardware shop that makes new digit plates for house numbers. If a house number is 195, for example, he has to create one plate for digit 1, one for digit 9, and one for digit 5. But, the orders are not always that simple. He may get orders to make digit plates, for example, for all houses in one side of a street.

Since making several plates of the same digit costs much less than making all digits for each house one by one, he wants to know, for a big order he receives, how many of each digit plate he has to make.

Input

The first number in the input line, t ($1 \leq t \leq 10$) is the number of orders. Following this, t orders are written in the input file. Each order starts with a line containing a street name, an arbitrary string of length at most 50 characters. The second line contains a single integer N ($1 \leq N \leq 10$), the number of sub-orders, followed by N lines of sub-orders. Sub-orders are of three kinds:

- A single house number: in this case, the sub-order line contains only a single integer n ($1 \leq n \leq 9999$);
- A series of house numbers: in this case, the sub-order line starts with a '`+`', followed by three integer numbers a, b, c ($1 \leq a, b, c \leq 9999$). This means that NarmakSung has to make plates for house numbers from a up to b with distance of c. That is, digit plates have to be made for house numbers a, $a + c, a + 2c, \ldots, b$. We assume that $a < b$, $b - a$ is a multiple of c, and $c \leq b - a$;
- A series of house numbers to be excluded: this kind of sub-orders specifies that a series of house numbers should not be made. In this case, the sub-order line starts with a '`-`', followed by three integer numbers with exactly the same conditions as in the previous case.

Note that if a house number is ordered more than once in two separate sub-orders, it is counted only once if it is not excluded at all (like number 100 in the second test case in the sample input). Also, if a house number is excluded somewhere in the test case, it cancels any order for that number, even if it appears later in the test case (like number 500 in the second sample). Note that it is possible to exclude some numbers that do not appear in other orders at all. In this case, these numbers are ignored (like 900 in the second sample).

Output

One set of output data is written for each input order consisting of 13 lines. Each set starts with one line containing the street name exactly as appeared in the input order. The next line must be of the form 'C `addresses`' where C is the total number of house numbers to be made. In the special case of $C = 1$, the output line should be '`1 address`'. The next 10 lines should be of the following form: Line i should contain the number of digit plates needed to be made for digit i. These 10 lines are on the format '`Make` X `digit` Y' where X is how many copies of digit Y they need to make. The last line 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. The output should be case-sensitive.

Sample

Input

```
2
Azadi Street
2
+ 101 103 2
275
Sharif Highway
3
100
- 500 900 100
+ 100 800 100
```

Output

```
Azadi Street
3 addresses
Make 2 digit 0
Make 3 digit 1
Make 1 digit 2
Make 1 digit 3
Make 0 digit 4
Make 1 digit 5
Make 0 digit 6
Make 1 digit 7
Make 0 digit 8
Make 0 digit 9
In total 9 digits
Sharif Highway
4 addresses
Make 8 digit 0
Make 1 digit 1
Make 1 digit 2
Make 1 digit 3
Make 1 digit 4
Make 0 digit 5
Make 0 digit 6
Make 0 digit 7
Make 0 digit 8
Make 0 digit 9
In total 12 digits
```

Problem C
Map Labeler

source: `maplabel.c` *or* `maplabel.cpp` *or* `maplabel.java`

Description

Map generation is a difficult task in cartography. A vital part of such task is automatic labeling of the cities in a map; where for each city there is text label to be attached to its location, so that no two labels overlap. In this problem, we are concerned with a simple case of automatic map labeling.

Assume that each city is a point on the plane, and its label is a text bounded in a square with edges parallel to x and y axis. The label of each city should be located such that the city point appears exactly in the middle of the top or bottom edges of the label. In a good labeling, the square labels are all of the same size, and no two labels overlap, although they may share one edge. Figure 1 depicts an example of a good labeling (the texts of the labels are not shown.)

Given the coordinates of all city points on the map as integer values, you are to find the maximum label size (an integer value) such that a good labeling exists for the map.

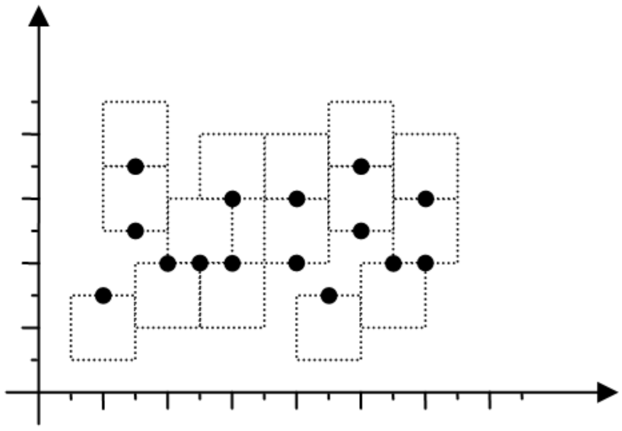

Figure 1: Example of labeling.

Input

The first line contains a single integer t ($1 \leq t \leq 10$), the number of test cases. Each test case starts with a line containing an integer m ($3 \leq m \leq 100$), the number of cities, followed by m lines of data each containing a pair of integers; the first integer (X) is the x and the second one (Y) is the y coordinates of one city on the map ($-10000 \leq X, Y \leq 10000$). Note that no two cities have the same (x, y) coordinates.

Output

The output will be one line per each test case containing the maximum possible label size (an integer value) for a good labeling.

Sample

Input	Output

```
1
6
1 1
2 3
3 2
4 4
10 4
2 5
```
	`2`

Problem D
IOI Photos

source: `photos.c` *or* `photos.cpp` *or* `photos.java`

Description

Shaborz, Hoidin, Alssein, and Ayan, members of the Olandican IOI team attended the Fall semester classes the same day they returned from IOI, Athens 2004. During their stay in Athens, they took several pictures in different places and occasions like Hydra island, opening ceremony, closing award ceremony, and city of Athens. But, being excited with their first university experience, they forgot about the pictures until the midterm recess, which has coincided with the ACM Regional Contest days. They now want to make prints of the pictures and each of them makes his own IOI album.

There are several negative rolls, and each contains photos of just a single place or occasion. There may be more than one roll, containing pictures from the same place or occasion. Each roll may have 36 negatives, numbered from 1 to 36. The team members and their friends want to order photo prints. Shaborz is to collect all orders and collects a fixed amount of money per each photo print. He makes a deal with a photo printing shop as follows and saves a good sum of money for himself. Shaborz pays S Rials for each single print, but printing all photos of a single role costs him R Rials, and printing all photos from all rolls in one order costs A Rials. Shaborz is provided with a list of orders, and you are to minimize the overall printing cost. Note that to have the minimum overall cost, Shaborz is allowed to print more photos than required.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case starts with one line containing four integers: N ($1 \leq N \leq 100$), the number of orders, S, R, and A, the costs of a single print, all prints from one roll, and all prints of all rolls respectively. Then follows N lines, each representing an order from one of the clients (team members and their friends). An order line contains a number of items separated by blank characters. Each item is of the form '*PlaceName*:*RollNo*:*FromPhoto*..*ToPhoto*'. *PlaceName* is the name of a place which is a string of at most 100 characters (case sensitive). *RollNo* specifies the desired roll among several rolls for the *PlaceName* and is between 1 and 10 inclusive. *FromPhoto* and *ToPhoto* are two numbers specifying the range of photos to be printed from the specified roll ($1 \leq FromPhoto \leq ToPhoto \leq 36$). You may assume there are at most 20 places. If there is only a single photo required from a roll, the format may be simplified as '*PlaceName*:*RollNo*:*PhotoNo*'. All costs are non-negative integers.

Output

For each test case, there should be one line containing one integer indicating the minimum cost for printing all photos of the original order set.

Sample

Input	**Output**

`1`
`2 15 100 400`
`Hydra:2:1..3 Athens:1:12`
`Delphi:1:4..5 Athens:3:20`
	`105`

Problem E
Dominoes

source: `dominoes.c` *or* `dominoes.cpp` *or* `dominoes.java`

Description

A domino is a flat, thumbsized tile, the face of which is divided into two squares, each left blank or bearing from one to six dots. There is a row of dominoes laid out on a table, like in Figure 2.

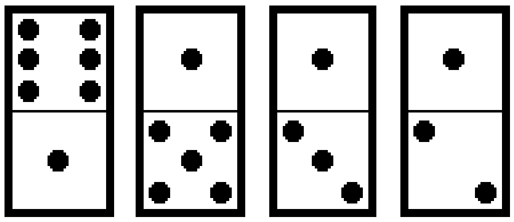

Figure 2: Row of dominoes laid out on a table.

The number of dots in the top line is $6 + 1 + 1 + 1 = 9$ and the number of dots in the bottom line is $1 + 5 + 3 + 2 = 11$. The gap between the top line and the bottom line is 2. The gap is the absolute value of difference between two sums.

Each domino can be turned by 180 degrees keeping its face always upwards.

What is the smallest number of turns needed to minimise the gap between the top line and the bottom line?

For Figure 2 it is sufficient to turn the last domino in the row in order to decrease the gap to 0. In this case the answer is 1.

Write a program that reads the number of dominoes and their descriptions from the input, computes the smallest number of turns needed to minimise the gap between the top line and the bottom line, and writes the result to the output.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Then follow t test cases.

The first line of a test case contains an integer n, $1 \leq n \leq 1000$. This is the number of dominoes laid out on the table. Each of the next n lines contains two integers a and b separated by a single space, $0 \leq a, b \leq 6$. The integers a and b written in the line $i + 1$ of the test case, $1 \leq i \leq n$, are the numbers of dots on the i-th domino in the row, respectively, in the top line and in the bottom one.

Output

For each test case, there should be one line in the output, containing the smallest number of turns needed to minimise the gap between the top line and the bottom line.

Sample

Input

```
2
4
6 1
1 5
1 3
1 2
10
1 6
6 0
0 6
0 4
6 4
0 4
6 3
3 2
2 1
5 1
```

Output

```
1
2
```

Problem F
Balanced Chemical Equations

source: `chemeq.c` *or* `chemeq.cpp` *or* `chemeq.java`

Description

One cumbersome problem in chemistry is the task of making the number of atoms balanced in a chemical equation. Our problem is concerned with this.

Chemists obey these rules when they present chemical equations:

1. Each element name is abbreviated by at most two letters. The first letter is always in upper-case and the second letter if exists, is a lower-case letter (e.g. Calcium is represented by Ca, Oxygen by O, and Chlorine by Cl).

2. Each molecule is composed of a number of atoms. To represent a molecule, we concatenate the abbreviated names of its composite atoms. For example, $NaCl$ represents Sodium Chloride. Each atom name may be followed by a frequency number. For example, Calcium Chloride $CaCl_2$ consists of one atom of Calcium and two atoms of Chlorine. If the frequency is not given, it is assumed to be 1 (so HCl is equivalent to H_1Cl_1). For the sake of simplicity, you may assume that the frequency of an occurrence of an atom is at most 9 (so we do not have $C_{11}H_{22}O_{11}$ in the problem input). Note that there may be several occurrences of the same atom in the molecule formula, like H atom in CH_3COOH.

3. In ordinary chemical reactions, a number of molecules combine and result in a number of other molecules. For example a well known sample of neutralization is:

$$2HCl + CaO_2H_2 \rightarrow CaCl_2 + 2H_2O$$

   This means two molecules of chlorohydric acid (HCl) with one molecule of Calcium Hydroxide (CaO_2H_2) results in one molecule of Calcium Chloride ($CaCl_2$) and two molecules of water.

4. In every chemical reaction, the total number of each atom in the right side of the equation equals the total number of that atom in the left side (that is why it is called an equation!).

Your task is to write a program to find appropriate coefficients $x_1, x_2, \ldots, x_M$ and $y_1, y_2, \ldots, y_N$ to balance an (imbalanced) equation like

$$A_1 + A_2 + A_3 + \cdots + A_M \rightarrow B_1 + B_2 + B_3 + \cdots + B_N$$

in the following way:

$$x_1A_1 + x_2A_2 + x_3A_3 + \cdots + x_MA_M \rightarrow y_1B_1 + y_2B_2 + y_3B_3 + \cdots + y_NB_N$$

Input

The first line contains an integer t ($1 \leq t \leq 10$), the number of test cases. Each test case consists of a single line containing an expression like:

$$A_1 + A_2 + A_3 + \ldots + A_M = B_1 + B_2 + B_3 + \cdots + B_N$$

Each A_i or B_i is a formula of a molecule according to the rules given in items 1 and 2.

The input equations are given in a way that if they can be balanced, x_i and y_i coefficients can be in the range of 1 to 9. There are less than 10 molecules and there are less than 10 different types of atoms, in a given equation. Additionally, you may assume molecules contain no more than 3 different kinds of atoms. You may also assume that there is no blank character in the input file, and the maximum length of the input lines is 200 characters.

Output

The output will be one line per test case containing the list of required coefficients, separated by blank characters, in the following order:

$$x_1\ x_2\ \ldots\ x_M\ y_1\ y_2\ \ldots\ y_N$$

The coefficients should be integers in the range of 1..9. Obviously, there may be more than one answer for a test case. In such situations, print the answer which minimizes the number:

$$x_1 x_2 \ldots x_M y_1 y_2 \ldots y_N$$

(This is an $(M + N)$-digit decimal number whose digits are x_i and y_i coefficients.) If the equation is impossible to balance, the output line should be '`IMPOSSIBLE`'.

Sample

Input	**Output**

`3`
`HCl+CaO2H2=CaCl2+H2O`
`HCl+H2SO4=NaCl`
`HCl+NaOH=NaCl+H2O`
	`2 1 1 2`
	`IMPOSSIBLE`
	`1 1 1 1`

Problem G
Points

source: `points.c` *or* `points.cpp` *or* `points.java`

Description

Let $p_1, p_2, \ldots, p_n$ be n points on the plane. We have m rules of form p_i *rel* p_j, each inform us that the relation *rel* holds among the locations of points p_i and p_j on the plane. For example, "p_i `NE` p_j" indicates that point p_j is located NorthEast of point p_i. There are eight different relations {`N`, `E`, `S`, `W`, `NE`, `NW`, `SE`, `SW`}, corresponding to the eight directions on the plane. Let (x_i, y_i) and (x_j, y_j) be the coordinates of p_i, and p_j respectively. Then p_i *rel* p_j exactly means one of the following, depending on the value of *rel*:

1. `N` stands for North. This means that $x_j = x_i$ and $y_j > y_i$,
2. `E` stands for East. This means that $x_j > x_i$ and $y_j = y_i$,
3. `S` stands for South. This means that $x_j = x_i$ and $y_j < y_i$,
4. `W` stands for West. This means that $x_j < x_i$ and $y_j = y_i$,
5. `NE` stands for NorthEast. This means that $x_j > x_i$ and $y_j > y_i$,
6. `NW` stands for NorthWest. This means that $x_j < x_i$ and $y_j > y_i$,
7. `SE` stands for SouthEast. This means that $x_j > x_i$ and $y_j < y_i$, and
8. `SW` stands for SouthWest. This means that $x_j < x_i$ and $y_j < y_i$.

The problem is to determine whether it possible to locate $p_1, p_2, \ldots, p_n$ on the plane so that all given rules are satisfied.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. The first line of each test case contains two integers n ($2 \leq n \leq 500$) which is the number of points and m ($1 \leq m \leq 10^4$) which is the number of rules. In each of the following m lines, there is one rule of the form i *rel* j which means that p_i has relation *rel* with p_j.

Output

The output contains one line per each test case containing one of the words '`POSSIBLE`' or '`IMPOSSIBLE`' indicating if the set of points in the test case can be located on the plane according to the given rules.

Sample

Input

```
2
3 2
1 N 2
2 N 1
6 6
1 E 2
1 E 3
2 N 4
3 NW 5
4 SW 6
6 NE 5
```

Output

```
IMPOSSIBLE
POSSIBLE
```

Problem H
Rotten Ropes

source: `ropes.c` *or* `ropes.cpp` *or* `ropes.java`

Description

Suppose we have n ropes of equal length and we want to use them to lift some heavy object. A tear-off weight t is associated to each rope, that is, if we try to lift an object, heavier than t with that rope, it will tear off. But we can fasten a number of ropes to the heavy object (in parallel), and lift it with all the fastened ropes. When using k ropes to lift a heavy object with weight w, we assume that each of the k ropes, regardless of its tear-off weight, is responsible for lifting a weight of w/k. However, if $w/k > t$ for some rope with tear-off weight of t, that rope will tear off. For example, three ropes with tear-off weights of 1, 10, and 15, when all three are fastened to an object, can not lift an object with weight more than 3, unless the weaker one tears-off. But the second rope, may lift by itself, an object with weight at most 10. Given the tear-off weights of n ropes, your task is to find the weight of the heaviest object that can be lifted by fastening a subset of the given ropes without any of them tearing off.

Input

The first line of the input file contains a single integer t ($1 \leq t \leq 10$), the number of test cases, followed by the input data for each test case. The first line of each test case contains a single integer n ($1 \leq n \leq 1000$) which is the number of ropes. Following the first line, there is a single line containing n integers between 1 and 10000 which are the tear-off weights of the ropes, separated by blank characters.

Output

Each line of the output file should contain a single number, which is the largest weight that can be lifted in the corresponding test case without tearing off any rope chosen.

Sample

Input	Output

2
3
10 1 15
2
10 15
	20
	20

Problem I
Food Cubes

source: `food.c` *or* `food.cpp` *or* `food.java`

Description

The spacemen in the space shuttle are waiting for the next escape window to return to the mother land Earth, where they are expected to fall somewhere in the deep blue waters of the Persian Gulf. Bored of waiting with nothing to do, they decide to play a game with their unit size food cubes. In the zero gravity environment of their spaceship, anything can stay motionless where it is placed. One spaceman places several food cubes in space such that there may be holes between cubes. Others, given the coordinates of the food cubes, should find the number of holes. A hole is a continuous empty space surrounded by food cubes in all six directions. You are to write a program to read the coordinates of each food cube and compute the number of holes.

Input

The first line of the input contains a single integer t ($1 \leq t \leq 20$) which is the number of test cases in the input. Each test case starts with an integer M, the number of food cubes. Each line i ($1 \leq i \leq M$) of the M following lines contains integers x_i, y_i and z_i, all between 1 and 100 inclusive, indicating the three coordinates of a food cube in the 3D space.

Output

For each test case, there is one line containing the number of holes.

Sample

Input

```
2
26
1 1 1
1 2 1
1 3 1
2 1 1
2 2 1
2 3 1
3 1 1
3 2 1
3 3 1
1 1 2
1 2 2
1 3 2
2 1 2
2 3 2
3 1 2
3 2 2
3 3 2
1 1 3
1 2 3
1 3 3
2 1 3
2 2 3
2 3 3
3 1 3
3 2 3
3 3 3
7
1 1 1
1 1 2
1 2 1
1 2 2
2 1 1
2 1 2
2 2 1
```

Output

```
1
0
```