Problem A: Laurel Creek

Laurel Creek is a perilous river that divides the campus into two halves and contains dangerous inhabitants such as geese and beavers. Your task in this problem is to find a way to cross the river without getting wet.

To do so, you will take advantage of several tree stumps in the middle of the river. A tree stump provides a safe place for you to stand as you ponder your next move. To get from one stump to another, you walk along logs that connect the stumps.

In cases where no log connects to the stump you wish to reach, all is not lost. You may pick up any log adjacent to the stump on which you are standing and put it down somewhere else so that it leads to the stump you wish to reach. In order for a log to be considered adjacent to a stump, it must be oriented in the appropriate direction; for example the log in `S-S` is adjacent to the two stumps, but the log in `S|S` is not considered adjacent to the two stumps.

Each tree stump is located at a point on a square grid. Two stumps are designated as the beginning and end point of the crossing. Any two stumps lying in the same row or column of the grid may be connected by a log. At any point in time, you may perform one of the following legal moves:

- Traverse a log adjacent to the tree stump you are standing on to the tree stump at the opposite end of the log.
- Pick up a log adjacent to the tree stump you are standing on. You may not hold more than one log at a time.
- Put down the log that you are holding so that it connects the stump you are standing on to some other stump. The log must be of precisely the right length to reach the other stump. The log must rest in the water: you may not use a log to connect two stumps if there is a third stump directly between them, or if the log would cross some other log already in the water.

Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers $1 <= r <= 15$ and $1 <= c <= 15$ specifying the number of rows and columns in the grid. Each of the next r lines of input contains c characters with the following meaning. The character `S` denotes a stump. The characters `B` and `E` denote the beginning and end stumps of the crossing, respectively. A consecutive sequence of `-` or `|` characters in a line denotes a single log whose length is proportional to the number of symbols. The character `.` denotes an empty grid point containing only water. There will never be more than fifteen stumps in the river.

Sample Input

```
1
7 11
....S......
....|......
B---S......
...........
...........
...........
....S.S...E
```

Output Specification

For each test case, output a line containing a single integer, the minimum number of moves in which the end stump can be reached from the initial configuration. If it is not possible to reach the end stump from the initial configuration, output a line containing the integer 0.

Output for Sample Input

```
10
```

Ondřej Lhoták

B Button Bashing

You recently acquired a new microwave, and noticed that it provides a large number of buttons to be able to quickly specify the time that the microwave should be running for. There are buttons both for adding time, and for subtracting time. You wonder how efficient you can be when entering cooking times: you want to minimize the number of required button presses.

The microwave can be running for at least 0 seconds, and at most 1 hour. If a button press would result in a cooking time of less than 0 seconds, the microwave will set the cooking time to 0 seconds. If a button press would result in a cooking time of more than 1 hour, the microwave will set the cooking time to 1 hour. Initially, the microwave will run for 0 seconds. There will always be a button adding at least 1 second to the cooking time.

Given the buttons that the microwave provides for entering cooking times, determine the least amount of button presses required to let the microwave run for a certain amount of time. If it is not possible to enter the desired cooking time precisely, determine the smallest achievable cooking time above the target, and the minimum number of button presses required for that cooking time, instead. The microwave does not allow to adjust the cooking time once it has started cooking.

EVER SINCE I HEARD THE SIMILE "AS NEGLECTED AS THE NINE BUTTON ON THE MICROWAVE" I'VE FOUND MYSELF ADJUSTING COOK TIMES.

source: xkcd.com/1103

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and t ($1 \leq n \leq 16$ and $0 \leq t \leq 3\,600$): the number of buttons available to change the cooking time, and the desired cooking time in seconds, respectively.
- one line with n space-separated integers b_i ($-3\,600 \leq b_i \leq 3\,600$): the number of seconds added to the cooking time when button i is pressed.

Output

Per test case:

- one line with two space-separated integers: the minimum number of button presses required to reach the required cooking time, and the minimum number of extra seconds that the microwave must be running for, respectively.

Sample in- and output

Input	Output
2 3 50 -10 10 60 1 50 20	2 0 3 10

Problem C: Virtual Friends

These days, you can do all sorts of things online. For example, you can use various websites to make virtual friends. For some people, growing their social network (their friends, their friends' friends, their friends' friends' friends, and so on), has become an addictive hobby. Just as some people collect stamps, other people collect virtual friends.

Your task is to observe the interactions on such a website and keep track of the size of each person's network.

Assume that every friendship is mutual. If Fred is Barney's friend, then Barney is also Fred's friend.

Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing an integer *F*, the number of friendships formed, which is no more than 100 000. Each of the following *F* lines contains the names of two people who have just become friends, separated by a space. A name is a string of 1 to 20 letters (uppercase or lowercase).

Sample Input

```
1
3
Fred Barney
Barney Betty
Betty Wilma
```

Output Specification

Whenever a friendship is formed, print a line containing one integer, the number of people in the social network of the two people who have just become friends.

Output for Sample Input

```
2
3
4
```

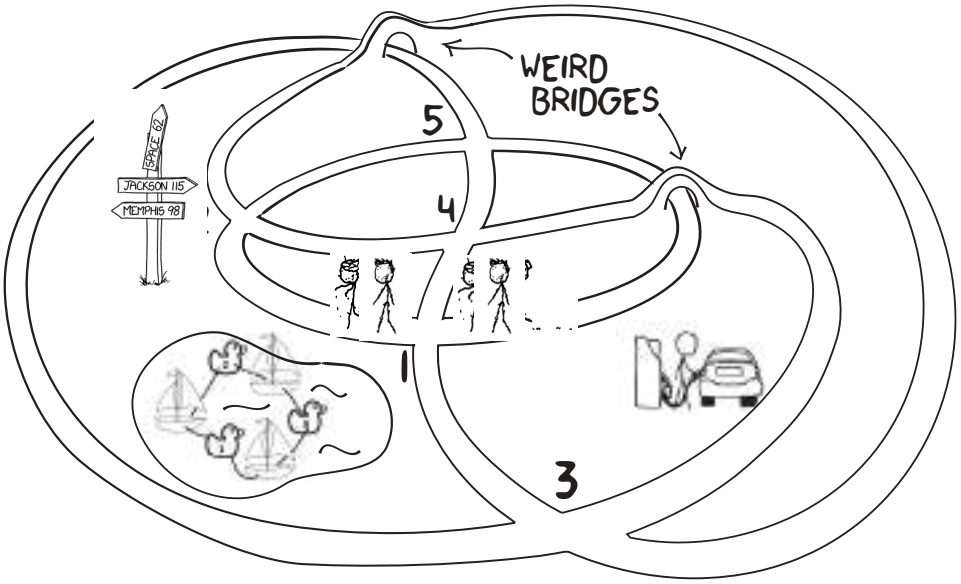
Ondřej Lhoták

D Dropping Directions

Your sports club has a rivaling sports club in the same city. They did some awful things to you and you want to get back at them. You have learned that they are planning on doing a 'drop': they drive people blindfolded to a city none of the participants know and tell them to find a specific place, the goal. They then have fun randomly walking through the city trying to find the goal.

You intend to spoil their fun thoroughly: you know that they promise a prize for whoever reaches the goal first, so the participants will use all available means to get to the goal. Indeed, you are fairly sure that if you set up official-looking signposts in that city in advance, they will probably follow them. You therefore decide to place signposts throughout the city so that no matter where the participants get dropped, they can follow the signposts to the goal; this takes out the element of 'randomly walking around' and therefore all the fun.

However, official-looking signposts are not cheap and attract a lot of attention, particularly from police officers. So you wish to minimize the number of signposts you have to place in the city. This may lead the participants to use a very slow detour, but they don't know the city anyway, so they won't find out.



City of 2nd and 3rd sample input. The 2nd input has the goal at intersection 5 and the 3rd at 4.

You get yourself a map of the city and start planning. You notice one nice aspect of the city: all intersections are cross-shaped, so it is easy to predict where participants will go to: they will just go to the opposite side of the intersection they arrive at. If a participant gets dropped at an intersection with a signpost, he or she will follow that sign; otherwise they go in an arbitrary direction until they hit a sign post. You know that participants never get dropped at the goal (that would be silly).

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with two space-separated integers n and g ($5 \leq n \leq 100\,000$ and $1 \leq g \leq n$): the number of intersections and the goal, respectively.
- n lines, each with four space-separated integers a, b, c and d ($1 \leq a, b, c, d \leq n$). The i-th line gives the intersections you end up at if you follow one of the roads adjacent to the i-th intersection; more precisely, participants who approach the i-th intersection coming from intersection a will continue towards intersection c and vice versa, while participants coming from intersection b will continue towards intersection d and vice versa.

Each intersection connects to four different other intersections.

Output

Per test case:

- one line with a single integer: the smallest number of signposts needed to ruin the fun.

Sample in- and output

Input	Output
4 5 5 2 3 4 5 3 4 5 1 4 5 1 2 5 1 2 3 1 2 3 4 5 5 5 3 2 4 4 5 3 1 1 5 2 4 1 3 5 2 1 3 2 4 5 4 5 3 2 4 4 5 3 1 1 5 2 4 1 3 5 2 1 3 2 4 5 5 5 3 2 4 4 5 3 1 1 5 2 4 1 5 2 3 1 3 2 4	0 0 1 1

Problem E: Logo

Logo is a programming language built around a turtle. Commands in the language cause the turtle to move. The turtle has a pen attached to it. As the turtle moves, it draw lines on the page. The turtle can be programmed to draw interesting pictures.

We are interested in making the turtle draw a picture, then return to the point that it started from. For example, we could give the turtle the following program:

```
fd 100 lt 120 fd 100 lt 120 fd 100
```

The command `fd` causes the turtle to move forward by the specified number of units. The command `lt` causes the turtle to turn left by the specified number of degrees. Thus the above commands cause the turtle to draw an equilateral triangle with sides 100 units long. Notice that after executing the commands, the turtle ends up in the same place as it started. The turtle understands two additional commands. The command `bk` causes the turtle to move backward by the specified number of units. The command `rt` causes the turtle to turn right by the specified number of degrees.

After executing many commands, the turtle can get lost, far away from its starting position. Your task is to determine the straight-line distance from the turtle's position at the end of its journey back to the position that it started from.

Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case starts with a line containing one integer, the number of commands to follow. The commands follow, one on each line. Each test case will contain no more than 1000 commands.

Sample Input

```
1
5
fd 100
lt 120
fd 100
lt 120
fd 100
```

Output Specification

For each test case, output a line containing a single integer, the distance rounded to the nearest unit.

Output for Sample Input

```
0
```

Ondřej Lhoták

F • Height Ordering

Mrs. Chambers always has her class line up in height order (shortest at the front of the line). Every September a new class of exactly 20 3rd graders arrive, all of different height. For the first few days it takes a long time to get the kids in height order, since no one knows where they should be in the line. Needless to say, there is quite a bit of jockeying around. This year Mrs. Chambers decided to try a new method to minimize this ordering chaos. One student would be selected to be the first person in line. Then, another student is selected and would find the *first* person in the line that is taller than him, and stand in front of that person, thereby causing all the students behind him to step back to make room. If there is no student that is taller, then he would stand at the end of the line. This process continues, one student at-a-time, until all the students are in line, at which point the students will be lined up in height order.

For this problem, you will write a program that calculates the total number of steps taken back during the ordering process for a given class of students.

Input

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K,*** followed by **20** non-negative unique integers separated by a single space. The 20 integers represent the height (in millimeters) of each student in the class.

Output

For each data set there is one line of output. The single output line consists of the data set number, ***K***, followed by a single space followed by total number of steps taken back.

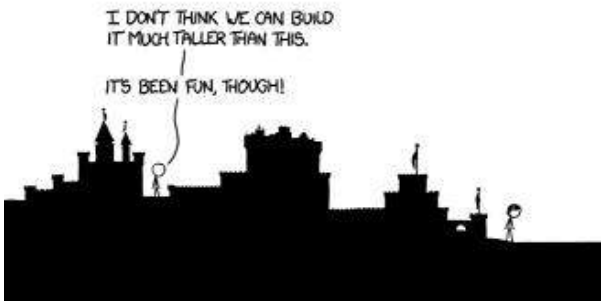
Sample Input
4 1 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 2 919 918 917 916 915 914 913 912 911 910 909 908 907 906 905 904 903 902 901 900 3 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 900 4 918 917 916 915 914 913 912 911 910 909 908 907 906 905 904 903 902 901 900 919

Sample Output
1 0 2 190 3 19 4 171

G Citadel Construction

The army wants to put up a new base in dangerous territory. The base will be surrounded by a citadel consisting of a number of straight walls. At every corner where two walls meet, there will be a watchtower. In order to enable efficient coordination in case of attack, the number of watchtowers should not exceed four.

After a detailed survey of the area, the army has concluded that not all locations are equally suitable for a watchtower: the ground needs to be firm enough to support the weight of the tower, and there should be enough visibility. They have selected a number of locations that meet the requirements.



source: xkcd.com/1190 (414)

Within these constraints, the army would like to build a base that is as large as possible. For this, they have come to you for help. Since the results of the survey are of course classified, you will not be given the possible locations directly; rather, you should write a program that can take them as input. Furthermore, they want the program to be able to handle multiple test cases in one go, so that they can hide the real data among lots of fake data.

For the computation of the area, you may assume that the watchtowers are infinitesimal in size and the walls infinitesimal in width.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($3 \leq n \leq 1\,000$): the number of locations that are suitable for a watchtower.
- n lines, each with two space-separated integers x and y ($-10\,000 \leq x, y \leq 10\,000$): the coordinates of each location.

All locations are distinct.

Output

Per test case:

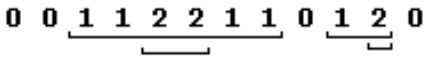
- one line with a single number: the largest possible area that the base can have. This number will be either an integer or a half-integer. If it is an integer, print that integer; if it is a half-integer, print the integer part followed by "`.5`". Trailing zeros are not allowed.

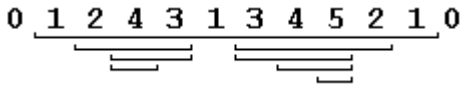
Sample in- and output

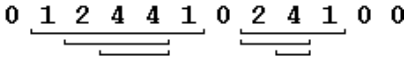
Input	Output
3 6 0 0 3 7 10 0 11 6 0 10 10 10 5 0 0 -2 -2 3 -2 0 1 0 3 10 3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3 2 3 8 4	100 12.5 31

H • Islands in the Data Stream

Given a sequence of integers **a1, a2, a3, …, a*n***, an *island* in the sequence is a contiguous subsequence for which each element is greater than the elements immediately before and after the subsequence. In the examples below, each island in the sequence has a bracket below it. The bracket for an island contained within another island is below the bracket of the containing island.







Write a program that takes as input a sequence of **12** non-negative integers and outputs the number of islands in the sequence.

Input

The first line of input contains a single integer ***P***, (**1** ≤ ***P*** ≤ **1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K,*** followed by **12** non-negative integers separated by a single space. The first and last integers in the sequence will be 0.

Output

For each data set there is one line of output. The single output line consists of the data set number, ***K***, followed by a single space followed by the number of islands in the sequence.

Sample Input	Sample Output
4 1 0 0 1 1 2 2 1 1 0 1 2 0 2 0 1 2 4 3 1 3 4 5 2 1 0 3 0 1 2 4 4 1 0 2 4 1 0 0 4 0 1 2 3 4 5 6 7 8 9 10 0	1 4 2 8 3 6 4 10

Problem I: Trainsorting

Erin is an engineer. She drives trains. She also arranges the cars within each train. She prefers to put the cars in decreasing order of weight, with the heaviest car at the front of the train.

Unfortunately, sorting train cars is not easy. One cannot simply pick up a car and place it somewhere else. It is impractical to insert a car within an existing train. A car may only be added to the beginning and end of the train.

Cars arrive at the train station in a predetermined order. When each car arrives, Erin can add it to the beginning or end of her train, or refuse to add it at all. The resulting train should be as long as possible, but the cars within it must be ordered by weight.

Given the weights of the cars in the order in which they arrive, what is the longest train that Erin can make?

Input Specification

The first line contains an integer 0 <= n <= 2000, the number of cars. Each of the following n lines contains a non-negative integer giving the weight of a car. No two cars have the same weight.

Sample Input

```
3
1
2
3
```

Output Specification

Output a single integer giving the number of cars in the longest train that can be made with the given restrictions.

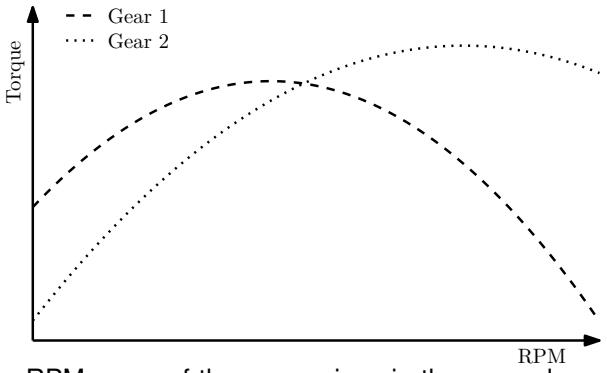
Output for Sample Input

```
3
```

Ondřej Lhoták

J Growling Gears

The *Best Acceleration Production Company* specializes in multi-gear engines. The performance of an engine in a certain gear, measured in the amount of torque produced, is not constant: the amount of torque depends on the RPM of the engine. This relationship can be described using a *torque-RPM curve*.



The torque-RPM curve of the gears given in the second sample input. The second gear can produce the highest torque.

For the latest line of engines, the torque-RPM curve of all gears in the engine is a parabola of the form $T = -aR^2 + bR + c$, where R is the RPM of the engine, and T is the resulting torque.

Given the parabolas describing all gears in an engine, determine the gear in which the highest torque is produced. The first gear is gear 1, the second gear is gear 2, etc. There will be only one gear that produces the highest torque: all test cases are such that the maximum torque is at least 1 higher than the maximum torque in all the other gears.

Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- one line with a single integer n ($1 \leq n \leq 10$): the number of gears in the engine.
- n lines, each with three space-separated integers a, b and c ($1 \leq a, b, c \leq 10\,000$): the parameters of the parabola $T = -aR^2 + bR + c$ describing the torque-RPM curve of each engine.

Output

Per test case:

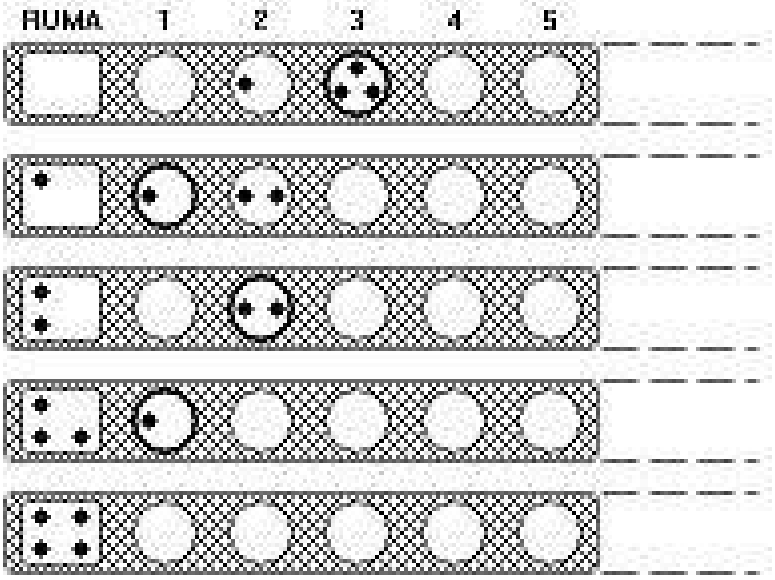
- one line with a single integer: the gear in which the maximum torque is generated.

Sample in- and output

Input	Output
3 1 1 4 2 2 3 126 1400 2 152 208 2 3 127 1400 2 154 208	1 2 2

K • **Mancala**

Mancala is a family of board games played around the world, sometimes called *sowing* games, or *count-and-capture* games, which describes the game play. One simple variant is a solitaire game called *Tchoukaillon* which was described by Véronique Gautheron. *Tchoukaillon* is played on a board with an arbitrary number of bins numbered **1**, **2**, **…,** containing `b[1], b[2], . . .` counters respectively and an extra empty bin called the *Roumba* on the left.



A single play consists on choosing a bin, ***n***, for which `b[n] = n` (indicated by the darker circles in the diagram) and distributing the counters one per bin to the bins to the left including the *Roumba* (getting the next diagram below in the figure above). If there is no bin where `b[n] = n`, then the board is a l*osing* board.

If there is a sequence of plays which takes the initial board distribution to one in which every counter is in the *Roumba*, the initial distribution is called a winnable board. In the example above, **0**,**1**,**3**,… is a winnable board (the "…" indicates all the bins to the right of **bin 3** contain **0**). For each total number of counters, there is a unique distribution of the counters to bins to make a winnable board for that total count (so **0**,**1**,**3**,… is the only winnable board with **4** counters).

Write a program which finds the winnable board for a total count input.

Input

The first line of input contains a single integer ***P***, (**1 ≤ *P* ≤ 1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, ***K,*** followed by a single space, followed by the total count ***N*** (**1** <= ***N*** <= **2000**) of the winnable board to be found.

Output

For each data set there will be multiple lines of output. The first line of output contains the data set number, ***K,*** followed by a single space, followed by the index of the last bin, ***B***, with a non-zero count. Input will be chosen so that ***B*** will be no more than 80. The first line of output for each dataset is followed by the bin counts **b[1], b[2], … , b[*B*]** ,10 per line separated by single spaces.

Sample Input	Sample Output
3 1 4 2 57 3 500	1 3 0 1 3 2 12 1 2 2 2 2 6 2 4 6 8 10 12 3 39 0 2 2 1 3 2 2 2 6 7 5 0 6 12 2 6 10 14 18 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39

Problem L: Dominos

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line. However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by hand to get the dominos falling again.

Your task is to determine, given the layout of some domino tiles, the minimum number of dominos that must be knocked down by hand in order for all of the dominos to fall.

Input Specification

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing two integers, each no larger than 100 000. The first integer *n* is the number of domino tiles and the second integer *m* is the number of lines to follow in the test case. The domino tiles are numbered from 1 to *n*. Each of the following lines contains two integers *x* and *y* indicating that if domino number *x* falls, it will cause domino number *y* to fall as well.

Sample Input

```
1
3 2
1 2
2 3
```

Output Specification

For each test case, output a line containing one integer, the minimum number of dominos that must be knocked over by hand in order for all the dominos to fall.

Output for Sample Input

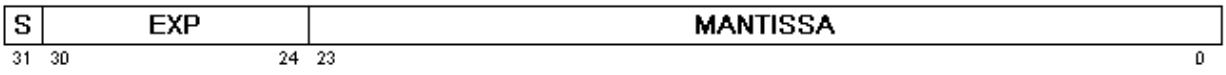
```
1
```

Ondřej Lhoták

M • Floating-Point Format Conversion

To help support a patent defense, we need to recover some experimental data that was stored as single precision floating point on a now-defunct Gould Power-Node mini-computer. The Gould used a base 16 floating-point format. We want to convert Gould floating point values, as much as possible, to single precision *IEEE* floating-point values.

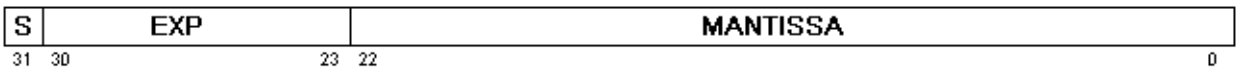
The Gould internal floating-point format has 1 sign bit, **S**, a 7-bit offset (base 16) exponent field, **E**, and a 24-bit (6 hex digits) hexadecimal mantissa. (Note that this means that up to 3 high bits of the mantissa may be zero.)



$$\text{value} = ((-1)^{S})((16)^{(EXP-64)})(MANTISSA / (2^{24}))$$

Floating-point zero is represented by 32 bits of 0.

The *IEEE* format has 1 sign bit, **S**, an 8-bit offset (base 2) exponent field, **E**, and a 24-bit mantissa, for which the high bit is (in normalized numbers) always 1 and not part of the 23 bits in the format.



If the exponent is not 255 and not 0, the value is a normalized floating point number,

$$\text{value} = ((-1)^{S})((2)^{(EXP-127)})(1 + (MANTISSA / (2^{23})))$$

If the exponent is 255 and the mantissa is 0, the value is plus or minus **infinity** (depending on the sign bit). If the exponent is 255 and the mantissa is not 0, it indicates special values that will not be used in this problem.

If the exponent is 0 and the mantissa is zero, the value is plus or minus zero (depending on the sign bit).

If the exponent is 0 and the mantissa is not zero, the value is a de-normalized floating-point number with:

$$\text{value} = ((-1)^{S})((2)^{(-126)})(MANTISSA / (2^{23}))$$

Write a program that takes as input a floating-point value in Gould format and outputs the value in IEEE format as follows:

If the value is zero return (plus) zero.

If the value is too large to be represented as a normalized floating-point value, return plus or minus infinity depending on the sign.

If the value is too small to be represented as a normalized floating-point value:
If it may be represented as a de-normalized value, return the de-normalized value.
Otherwise, return plus or minus zero, depending on the sign.

In all other cases, return the normalized value.

If there are less significant bits than required for IEEE floating-point, extend with 0 bits.

If there are more significant bits than required for IEEE floating-point, truncate the extra bits.

Input

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, K, followed by 8 hex digits (**0-9,A-F**) of the Gould floating-point value.

Output

For each data set there is one line of output. The single output line consists of the data set number, K, followed by a single space followed by the 8 hex digits (**0-9,A-F**) of the corresponding (as described above) *IEEE* floating point value.

Sample Input	Sample Output
4 1 41200000 2 E0FFFFFE 3 E11FFFFF 4 88888888	1 40000000 2 FF7FFFFE 3 FF800000 4 80000000

N • Happy Happy Prime Prime

RILEY VASHTEE: [*reading from display*] Find the next number in the sequence:

313 331 367 …? What?

THE DOCTOR: **379.**

MARTHA JONES: What?

THE DOCTOR: It's a sequence of happy primes – **379**.

MARTHA JONES: Happy *what?*

THE DOCTOR: Any number that reduces to one when you take the sum of the square of its digits and continue iterating it until it yields **1** is a happy number. Any number that doesn't, isn't. A *happy prime* is both happy and prime.

THE DOCTOR: I dunno, talk about *dumbing down.* Don't they teach recreational mathematics anymore?

Excerpted from "*Dr. Who*", Episode 42 (2007).

The number 7 is certainly prime. But is it happy?

$$\begin{aligned} 7 &\rightarrow 7^2 = 49 \\ 49 &\rightarrow 4^2 + 9^2 = 97 \\ 97 &\rightarrow 9^2 + 7^2 = 130 \\ 130 &\rightarrow 1^2 + 3^2 = 10 \\ 10 &\rightarrow 1^2 + 0^2 = 1 \end{aligned}$$

It is happy ☺. As it happens, 7 is the smallest happy prime. Please note that for the purposes of this problem, 1 is *not* prime.

For this problem you will write a program to determine if a number is a *happy prime*.

Input

The first line of input contains a single integer P, ($1 \le P \le 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, K, followed by the happy prime candidate, m, ($1 <= m <= 10000$).

Output

For each data set there is a single line of output. The single output line consists of the data set number, K, followed by a single space followed by the candidate, m, followed by a single space, followed by **YES** or **NO**, indicating whether m is a happy prime.

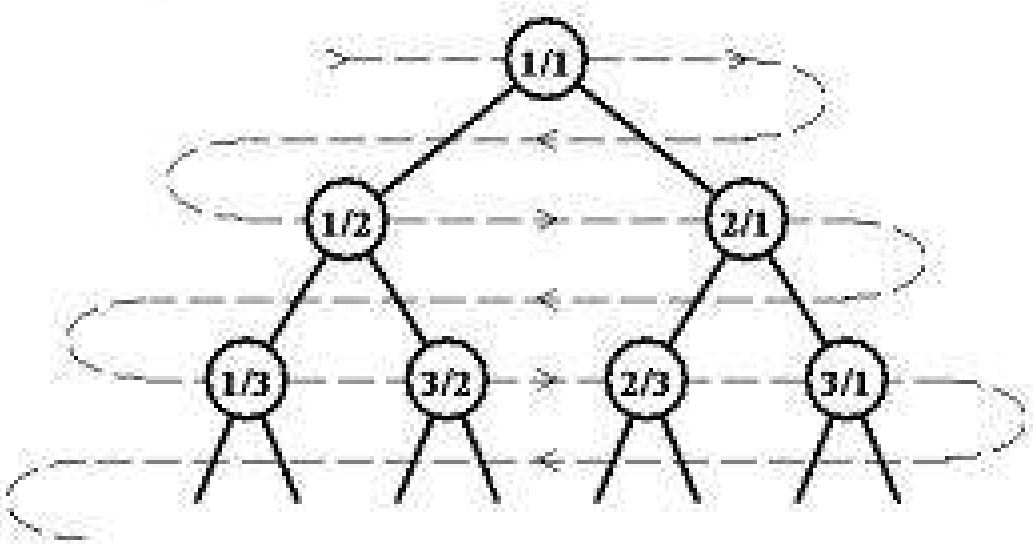
Sample Input	Sample Output
4 1 1 2 7 3 383 4 1000	1 1 NO 2 7 YES 3 383 YES 4 1000 NO

O • A Rational Sequence

An infinite full binary tree labeled by positive rational numbers is defined by:

- The label of the root is `1/1`.
- The left child of label `p/q` is `p/(p+q)`.
- The right child of label `p/q` is `(p+q)/q`.

The top of the tree is shown in the following figure:



A rational sequence is defined by doing a level order (breadth first) traversal of the tree (indicated by the light dashed line). So that:

`F(1) = 1/1, F(2) = 1/2, F(3) = 2/1, F(4) = 1/3, F(5) = 3/2, F(6) = 2/3, …`

Write a program which takes as input a rational number, `p/q`, in lowest terms and finds the next rational number in the sequence. That is, if `F(n) = p/q`, then the result is `F(n+1)`.

Input

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input. It contains the data set number, K, which is then followed by a space, then the numerator of the fraction, **p**, followed immediately by a forward slash (/), followed immediately by the denominator of the fraction, **q**. Both **p** and **q** will be relatively prime and 0 <= **p, q** <= 2147483647.

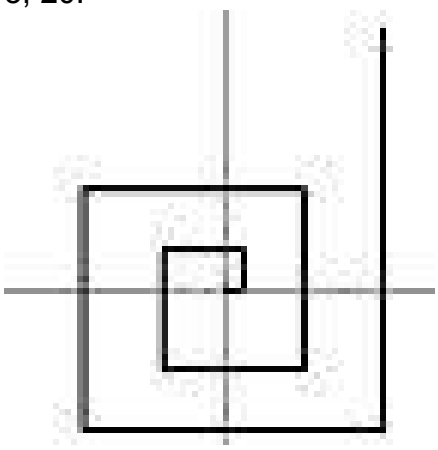
Output

For each data set there is a single line of output. It contains the data set number, ***K***, followed by a single space which is then followed by the numerator of the fraction, followed immediately by a forward slash ('/') followed immediately by the denominator of the fraction. Inputs will be chosen such that neither the numerator nor the denominator will overflow a 32-bit integer.

Sample Input	Sample Output
5 1 1/1 2 1/3 3 5/2 4 2178309/1346269 5 1/10000000	1 1/2 2 3/2 3 2/5 4 1346269/1860498 5 10000000/9999999

P • Growing Rectangular Spiral

A growing rectangular spiral is a connected sequence of straight-line segments starting at the origin. The first segment goes right (positive **x** direction). The next segment goes up (positive **y** direction). The next segment goes left (negative **x** direction). The next segment goes down (negative **y** direction) and the sequence of directions repeats. Each segment has integer length and each segment is at least one unit longer than the previous segment. In the spiral below, the segment lengths are 1, 2, 4, 6, 7, 9, 11, 12, 15, 20.



Write a program to determine the shortest growing rectangular spiral (in total length) that ends at a given integer point **(x,y)** in the first quadrant or determine that there is no such spiral.

Input

The first line of input contains a single integer ***P***, (**1 ≤ *P* ≤ 1000**), which is the number of data sets that follow. Each data set should be processed identically and independently.

Each data set consists of a single line of input consisting of three space separated decimal integers. The first integer is the data set number. The next two integers are the **x** and **y** coordinates of the desired end point (**1 <= x <= 10000, 1 <= y <= 10000**).

Output

For each data set there is a single line of output. If there is no spiral solution, the line consists of the data set number, a single space and "**NO PATH**" (without the quotes). If there is a solution, the line consists of the data set number, a single space, the number of segments in the solution, a single space, followed by the lengths of the segments in order, separated by single spaces. The input data will be chosen so that no path requires more than **22** segments.

Sample Input	Sample Output
3 1 1 1 2 3 5 3 8 4	1 NO PATH 2 2 3 5 3 6 1 2 3 9 10 11