Antisort

Usually, in programming contests the problem statement tells you what to do. We find that boring.

In this task we do the opposite. We will tell you something you **must not** do:

You must not sort the given sequence.

Problem specification

Given is a sequence S of distinct integers. Rearrange the sequence in any way. The only condition is that the resulting sequence must not be sorted – neither in ascending nor in descending order.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of two lines. The first line contains a number N (between 3 and 1000, inclusive) giving the length of the sequence. The second line contains N space separated numbers that form the sequence. No two of these N numbers are equal, and all of these numbers fit into a 32-bit signed integer variable.

Output specification

The output format is the same as the input format, with one exception: The whitespaces may be arbitrary. Just make sure that every two tokens in the output are separated by at least one whitespace.

Each sequence in the output must contain the same numbers as the corresponding input sequence, and they must not be in sorted order.

Example

input

```
2

5
1 2 3 4 5

8
3 1 4 47 5 9 2 6
```

output

```
2

5
5 1 4 3 2

8
3 1 4 47
5 9 2 6
```

Broadway

The Manhattan in the New York City has really a nice topology. So nice, it is often idealized to a rectangular grid. If you want to go from corner $A = (A_x, A_y)$ to corner $B = (B_x, B_y)$, the shortest way has length $|A_x - B_x| + |A_y - B_y|$. Or so they told you in school.

The truth is, the correct definition of a Manhattan metric has to involve the Broadway – a road that leads across the neatly aligned system of streets and avenues. In this problem we finally correct this horrible mistake made by the mathematical community.

Problem specification

Given the two corners $A = (A_x, A_y), B = (B_x, B_y)$ and three rational numbers P, Q, R that describe the Broadway, your task is to find the length of the shortest path between points A and B.

The road network consists of the following roads:

- For each integer Z, there is an avenue described by the equation $x = Z$.
- For each integer Z, there is a street described by the equation $y = Z$.
- The Broadway is described by the equation $Px + Qy = R$.

When moving from A to B, we can only move along the roads and change roads at intersections.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of one line containing seven numbers:
four integers A_x, A_y, B_x, B_y specifying the points $A = (A_x, A_y)$ and $B = (B_x, B_y)$,
and three rational numbers P, Q, R specifying the Broadway as explained above.

Output specification

For each test case output a single line containing the length of the shortest path from A to B. Output a sufficient number of decimal places. Your output will be judged as correct if it has an absolute or relative error at most 10^{-9}.

Example

input

```
2

2 0  -1 1  1.0 1.0 1.0

-2 3  4 -1  1.0 -0.1 0.47
```

output

```
3.414213562373
10
```

Cells

Tim loves spreadsheets. Everything he does on a computer, he does in a spreadsheet. Track his expenses? Create a spreadsheet! Decide which car to buy? Create a spreadsheet to compare them! Make an inventory of his games? Create a spreadsheet! Decide which girl he loves most? ...

Unfortunately his spreadsheet software just crashed and he needs some of the data right now and does not have the time to install a competing office suit.

Problem specification

Given the formulas used in the cells of a spreadsheet, calculate the values of all the cells.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts a single number N giving the number of expression. Each of the following N lines contains a single cell formula of the form "`CELL = EXPRESSION`", where `CELL` is the name of the cell and `EXPRESSION` is a mathematical expression consisting of cell names and the operators `+`, `-`, `*` and `/`. A cell name is a non-empty sequence of letters followed by a positive integer.

Each test case is correct: there are no cycles, and all cells referenced in expressions have definitions.

Evaluating expressions

When evaluating an expression, usual priorities apply: first we evaluate all `*` and `/` (left to right), and only then all `+` and `-` (again, left to right).

You may assume that the expressions are such that when evaluating the expression in correct order, the result and also all intermediate values will fit into 32-bit signed integer variables.

The operator `/` represents integer division which is always rounded **down**. The dividend will always be non-negative and the divisor will always be positive.

Output specification

For each test case output the calculated values of cells, one per line, in the form "`CELL = VALUE`". The rows in the output should be ordered alphabetically. (To compare two rows, take a look at the first character where they differ. The one with a smaller ASCII value goes first.)

Optionally, output a blank line between test cases.

Example

input

```
2

3
A47 = 5 + ZZ22
ZZ22 = 3
A9 = 13 + A47 * ZZ22

2
A1 = 4 / 7 + 4 / 7
B2 = 3 * 3 / 7
```

output

```
A47 = 8
A9 = 37
ZZ22 = 3

A1 = 0
B2 = 1
```

Dragon Slayer

You probably heard the story a thousand times: In some place far away, at some time long ago there was a kingdom ruled by a good king. This king had a daughter – a pretty princess. One day an evil dragon abducted the princess. A prince from some other kingdom decided to kill the dragon and save her. There was a fight, the dragon died, the princess was saved, and everybody lived happily ever after.

Of course, the story tellers have no idea how the fight with a dragon really looks like, so they make up a fake story where the prince uses a combination of brute force, expert sword handling and clever tricks to kill the dragon.

In reality, killing a dragon is a purely mathematical problem. The dragon has a variable (but always non-negative) number of heads. The only way to kill it is to reduce the number of its heads to zero.

The number of heads can only be changed by using a magic sword. Each magic sword is described by two parameters: c and g. Each time someone swings the magic sword, he must cut off **exactly** c heads. Afterwards, if the dragon is still alive (its number of heads is positive), it grows g new heads.

The sword cannot be used if there are less that c heads to cut. Note that the prince has a head. Thus he can kill a dragon with exactly $c - 1$ heads, but dies when doing so.

Problem specification

Our prince has found **two magic swords**: one with parameters c_1 and g_1, the other with parameters c_2 and g_2. The dragon has N heads.

Given the numbers N, c_1, g_1, c_2 and g_2, compute whether the prince can kill the dragon, and if so, whether he can survive the fight.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of one line containing the numbers N, c_1, g_1, c_2 and g_2.

Output specification

For each test case output a single line containing one word: "PRINCE" if the prince can win the fight (i.e., kill the dragon and survive), "DRAW" if he can kill the dragon but has to die in order to do so, or "DRAGON" if the dragon cannot be killed.

Example

input

```
3

20 7 1 8 5

3 4 1 2 2

100 102 0 103 0
```

output

```
PRINCE
DRAW
DRAGON
```

In the first case, the prince can kill the dragon e.g. by using the swords in the order first, first, second.

In the second test case the second sword has no effect. As soon as the prince uses the first sword, both he and the dragon die.

In the third test case the prince cannot use any of the two swords.

Eyjafjallajökull

Sometimes it's hard to see the forest, because all the trees get in your way. Other times, it's hard to see the volcano, because of all the ash in the air.

Problem specification

The world is represented by a rectangle of $M \times N$ cells. There is exactly one volcano somewhere in the world. The volcano occupies one of the cells. Your task is to find this cell.

Note that there are two separate data sets. The easy data set and the hard data set are completely independent and use two different maps. You have to solve each of them separately.

In the beginning, there is no ash anywhere. Our volcano erupts in rounds that correspond to your submits, and there is a changing wind which then distributes the ash over the world.

In each your submit you can check K locations of the map to see if there is currently any ash present or not. These probes are binary – it will report ash presence if and only if the cell contains at least one unit of ash.

Each round consists of the following steps, in order.

1. You submit a list of probes.
2. The volcano releases 100 units of new ash into the air at its location (no wind applied yet).
3. Your probes are evaluated.
4. A new wind is calculated and applied to the whole map. (In each round the current wind is the same everywhere on the map.)

The answer you'll receive for your submit will contain all probe results and a description of the wind that was generated in step 4.

Wind model

The wind is specified by a 5×5 matrix called the *wind matrix*. Rows and columns of a wind matrix are labeled from -2 to 2.

A wind matrix specifies how to distribute the ash from any given cell into its surroundings. In each wind matrix the numbers are non-negative integers that sum up to 100, and they should be interpreted as percentages.

Formally, suppose that there are a units of ash in a given cell (x, y) before the wind blows. Let W be a wind matrix. After the wind described by W blows, the ash from (x, y) will be redistributed into cells (x', y') with $x-2 \leq x' \leq x+2$ and $y-2 \leq y' \leq y+2$. More precisely, the cell $(x+d_x, y+d_y)$ will receive $\lfloor a \cdot W_{d_x,d_y}/100 \rfloor$ units of ash from this cell. Any ash that leaves the map boundary in this way is gone forever. Also note that (due to rounding) the total amount of ash the cells receive may be less than a.

It is important to note that the ash from all cells is being redistributed at the same time. (The process resembles a blur operation in a graphics editor.)

Input specification

The input file contains 3 integers: M, N and K. The map size is $M \times N$ cells. In each submit, you may specify up to K probe locations.

The volcano will be placed in a cell such that in any direction there are at least two other cells before we reach the map boundary.

Output specification

In each submit you can specify up to K cell locations x_i, y_i ($0 \le x_i < M$, $0 \le y_i < N$). All values should be separated by whitespace. In case one of your probes matches the volcano location, the game ends and the data set is correctly solved.

Otherwise, you will receive a "`WRONG ANSWER`" message with the following information:
– the current round number,
– ash presence values for all your probes (0 = no ash present, 1 = some ash present),
– and a new wind matrix.

Submission limit

Please note that the submission limit for each data set is 10 submits, as is the case for any other task.

Be careful about the submissions you make. Try not to waste any, and always double-check whether the one you are about to send is correct.

Example submission (in the middle of a game)

input

```
1 3
4 9
```

output

```
Round:  3

Probe [1,3]:  0
Probe [4,9]:  1

Wind:
0 0 5 5 0
0 5 30 10 0
0 0 20 15 0
0 0 10 0 0
0 0 0 0 0
```

Flawed Code

Michael is teaching an algorithms class. In the last lecture he gave his students the following problem:

> Given are several cubes of various sizes. You can make towers out of the cubes by placing them one atop another, in any order. Your task is to find the *largest possible* height H such that is is possible to build *two separate towers* of height H.
>
> For example, if the cube sizes are (8,6,3,3,3,2), then the solution is 11:
> we can build one tower as 8+3, the other as 6+3+2, and a cube of size 3 remains unused.

This task proved to be too difficult for his students, so each of them just implemented some clever hack that sometimes worked and sometimes failed. More precisely:

- Alla implemented a brute force solution. She sorts the cubes in descending order. If there are more than 15, she just takes the 15 largest cubes. Then she tries all possible ways how to build two towers and finds the best one in which the towers are of equal height.
- Bob finds and outputs the best solution that only leaves at most two cubes unused.

  To do so, he used dynamic programming to implement a function that takes a set of cubes and checks whether it is possible to build two equal towers using *all of them*. He then calls this function multiple times, each time with a different subset of the given set of cubes.
- Chermi loves greedy algorithms. His program finds the largest X for which his greedy algorithm manages to build two towers of height X each.

  His algorithm to build two towers of height X is simple: He first sorts the cubes in descending order, and then processes them one by one. Each time he checks whether he can add the cube to the currently smaller tower without exceeding X, and if he can, he does so.
- Dominika's solution is perhaps the most clever one. She used dynamic programming. For each H she computed the number of ways in which a tower of size H can be assembled. The answer she outputs is the largest X such that the tower of height $2X$ can be assembled, and the tower of height X can be assembled in at least two different ways.

Problem specification

You are given the four programs made by Michael's students (for your convenience, each of them is implemented in several languages). Your goal is to help Michael show the students their errors by providing inputs for which the students' solutions fail.

Input specification

The input file contains implementations of the four students' programs in C++, Pascal, and Python.

Output specification

As the output for the **easy** data set F1, output 4 lines, containing 2 valid test cases. Each test case consists of two lines. The first line contains the number N of cubes ($1 \leq N \leq 100$). The second line contains N space-separated integers – the cube sizes. The sum of all cube sizes must not exceed 10 000.

The submission will be judged as correct if it breaks at least three of our solutions. In other words, at most one of our four solutions can give the correct answer for both your test cases.

As the output for the **hard** data set F2, output 2 lines containing a single valid test case.

The submission will be judged as correct if it breaks all four solutions. That is, none of the four solutions should give the correct answer for your test case.

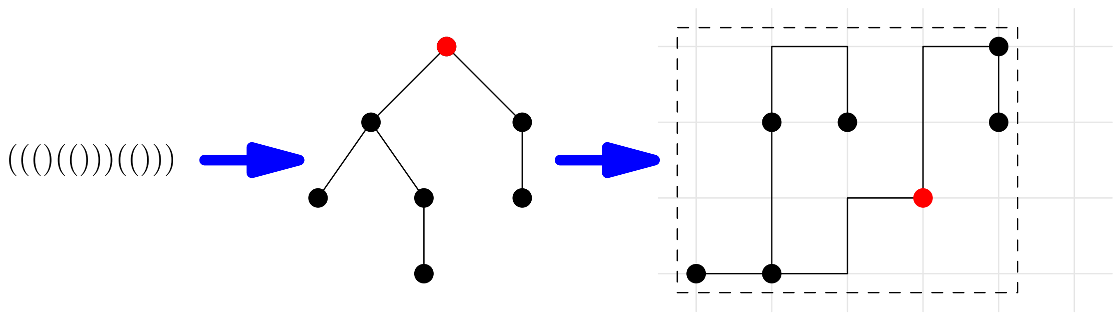
Grid is good

Little Gordon enjoys nature and especially trees. He would like to sketch some of his most favourite ones. However, it seems that he doesn't have enough grid paper – and that's where your help is needed...

Problem specification

You are given a rooted binary tree and a number K. Pick a rectangle on the grid with sides parallel to the coordinate axes. The chosen rectangle may contain at most K grid points (including those on its perimeter). Draw the tree into the rectangle.

You have to place each node onto one of the grid points in your chosen rectangle. Different nodes must be placed onto different points. You then have to draw each of the edges. Each edge must be drawn as a polyline that lies entirely on the grid and does not exit the rectangle. No two edges may share a common point other than an endpoint they have in common.



Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of two lines. The first one contains the integer K, the second one a string of parentheses that encodes the tree.

An empty tree is encoded as an empty string. A tree consisting of a root node and its two subtrees is denoted `(LR)`, where `L` and `R` are the codes of those two subtrees. Note that a leaf has two empty subtrees, hence each leaf is denoted by the string `()`.

The trees in the hard data set are identical to those in the easy data set. The data sets differ only in the amount of grid points you can use (the number K).

Output specification

For each test case output a single line with two integers r and c: the number of rows and the number of columns of grid points your chosen rectangle contains. Note that rc must not exceed K.

Afterwards, output $2r - 1$ rows, each containing $2c - 1$ characters: an ASCII art drawing of the tree. In the drawing, grid points correspond to characters whose both coordinates are even (indexed from 0).

The root node should be marked as `H`. For each other vertex: if it is the first child of its parent, label it `L`, otherwise label it `R`. The edges are drawn using `-`, `|` and `+` symbols (ASCII 45, 124, and 43; follow the example output). All the remaining characters should be `.` (ASCII 46).

Any valid drawing will be accepted.

Example

input

```
2

4
(()())

36
(((()(()))(())))
```

In the second test case the chosen rectangle contains 20 grid points, which is less than the limit (which is 36).

The second test case and its solution that corresponds to this ASCII art drawing are shown in the picture in the problem statement.

output

```
2 2
H-R
|..
L..

4 5
..+-+.+-R
..|.|.|.|
..R.L.+.L
..|...|..
..+.+-H..
..|.|....
L-L-+....
```

Hash

When dealing with large files, hashing is a nice way how to reduce the file into a small *fingerprint* – a short string that somehow represents the contents of the entire file.

Hashing has multiple uses, for example it gives us a nice integrity check when transfering the file. It is also used in most digital signatures – as signing is usually slower than hashing, you first hash the file, and then sign the fingerprint.

Most of the hash functions have the property that even if just one byte of the file changes, we have to process the entire file again in order to find the new hash value.

In this task we investigate some options how to design a hash function that would allow us to update the hash value quickly after we make a change to the hashed file. We will use the following approach:

1. We split the file into blocks $w_1, \dots, w_n$ of size 256 bytes (for simplicity we will assume that the length of the file is divisible by 256).
2. We calculate the hash of each block: $H_i = H(w_i)$, where H is some cryptographic hash function whose output is m bits long. We will use the well-known MD5 hash function with $m = 128$ bits.
3. We merge the block hashes together. This can be done in two simple ways. We can either use bitwise xor (so the final hash would be $X = H_1 \oplus H_2 \oplus \dots \oplus H_n$), or we can add the individual hashes as numbers, computing modulo 2^m (so the final hash would be $A = (H_1 + H_2 + \dots + H_n) \bmod 2^m$). We will call the two approaches XOR-HASH and ADD-HASH, respectively.

Problem specification

Invert the hash functions described in the problem statement. That is, both for XOR-HASH and for ADD-HASH you should provide a file with a prescribed hash value.

Note that while there are known attacks to find collisions in MD5, so far it is not known how to invert this function efficiently.

Input specification

You are given a sample implementation of the MD5 hash function along with a testing program and example output files for a team with TIS 12345678901234567890.

Output specification

As the output for the **easy** data set H1, output the hex dump of a file which has $n = 256$ blocks (i.e., its size is 65536 bytes). After calculating the XOR-HASH of this file, the hexadecimal notation of the result must be `00000 00000 00XXX XXXXX XXXXX XXXXX XX`, where the string of 20 `X`s should be replaced by your TIS.

As the output for the **hard** data set H2, output the hex dump of a file which has $n = 128$ blocks (i.e., its size is 32768 bytes). The ADD-HASH of this file must be `00000 00000 00XXX XXXXX XXXXX XXXXX XX`, where the string of 20 `X`s should be replaced by your TIS.

The hex dump of a B-byte file should consist of B pairs of hexadecimal digits separated by any whitespace. For example, if the first three bytes of your file have ASCII values 97, 32, 10, the hex dump of this file would start "`61 20 0a`".

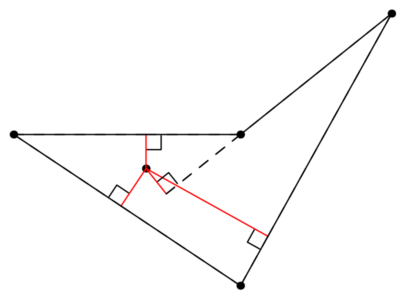
Instead of the provided MD5 implementation, you should be able to use any software that computes MD5 hashes of files. If your operating system contains the `od` application (octal dump, a part of GNU coreutils), you can easily create the hex dump of a file using the command `od -v -A n -t x1`.

Inside job

Jan is throwing darts into a polygon painted on the wall. His aim is really lousy. We will assume that the points he hits are uniformly distributed inside the polygon.

(Formally, let A be the area of the polygon. Given any part of the polygon with area B, the probability that Jan hits the given part is B/A.)

Each time Jan throws a dart, Monika measures the altitudes from the point where the dart landed onto all sides of the polygon. In other words, for each side of the polygon she measures the distance between the dart and the line that contains the given side. She then adds all the distances together and announces the result.


Monika computes the total length of the red lines in the image.

Problem specification

Given is the polygon. Compute the expected value of the number Monika will calculate.

(If you do not know what an expected value is, imagine it as the average of very many attempts.)

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a line containing the number of vertices N. Each of the next N lines contains two integers: the coordinates of one point. You may assume that $N \leq 500$ and that all polygons are simple: the boundary of a polygon never touches or intersects itself.

The polygons in the easy input are special: They are convex, N is even, and each pair of opposite sides is parallel.

Output specification

For each test case output a single line with the expected value of the sum of all distances. The number must be printed with sufficiently many decimal places (we recommend 10 or more). Answers that have an absolute or relative error at most 10^{-9} will be accepted.

Example

input

```
2

3
0 0
3 0
3 4

3
4 2
6 1
7 3
```

output

```
3.133333333333
2.017758261695
```

Jimmy the Acting Teacher

Jimmy is an acting teacher. He wants to introduce a new acting exercise to his class. There are exactly N men and N women taking his course. There are N tables in the room. Each table is assigned a dialogue for a man and woman. The dialogues are different at different tables. Jimmy wants all students to try every dialogue exactly once. He also wants every man to play with every woman, so that he can see which actors should play together in a play.

Jimmy wants to keep things as simple as possible. First, he took a black marker and labelled the tables from 1 to N. Then he took a blue marker and a red marker and wrote a blue number and a red number onto each table. Whenever a couple finishes the dialogue, the man reads the blue number on their table and goes to the table with that number, and the woman does the same with the red number.

Each actor chooses one table at the beginning of the class and will start by playing the dialogue at the chosen table. Each dialogue takes 5 minutes. After the 5 minutes are over, all students move simultaneously according to the blue and red numbers. All movement happens in an instant.

Problem specification

Finding the right 2 numbers for each table seems to be a very difficult task. Please help Jimmy. For a given N, find one way how the blue and red numbers may look like.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of a single line containing the integer N denoting the size of the class.

Output specification

For each test case output a single line with word "`Impossible`" if it is impossible to find the right 2 numbers for every table. Otherwise output 2 lines. The first line will contain a space-separated list of blue numbers, the second line will contain a space-separated list of red numbers. (The i-th number in each list is the one written on table i.) Output a blank line between test cases.

Example

input

```
3

2

3

5
```

output

```
Impossible

2 3 1
3 1 2

2 3 4 5 1
5 1 2 3 4
```

In the first test case there is no solution: after the first dialogue both actors at table 1 must move to a different table in order not to play the same dialogue twice. But then they will face each other again.

In the second test case in the first round we have pairs $(1,1),(2,2),(3,3)$ playing together (assuming that we numbered both men and women 1, 2, 3 according to their starting table). In the second round pairs $(3,2),(1,3),(2,1)$ play together. Finally, in the third round pairs $(2,3),(3,1),(1,2)$ play together. Note that everyone played with all possible partners and visited all tables.

Klingon High Council Training

Tired of being seen as the endless supply of brute force for the United Federation of Planets, the Klingon High Council has decided to start a training program aimed on spaceship battle tactics. The main part of the program is a game that is played by two of the trainees.

The game is played using a fleet of N spaceships. The space is divided into a cube grid of sectors. Each sector is uniquely identified by its three integer coordinates. Each sector may contain multiple spaceships. During the game the spaceships are traveling between sectors in order to reach a target planet that is located in the sector $(0, 0, 0)$. When all the ships reach this sector, the game is over and the player who finished the maneuver is decorated as the winner.

Each turn of the game looks as follows: The player whose turn it is must first **contact** between 1 and K ships, and then **order** each of the contacted ships to move. The turn ends when all ships finish executing their orders.

Different ships may receive different orders. Each ship must be ordered to move into a sector that is strictly closer to the target planet than its current sector. (When measuring the distance between sectors, we measure the Euclidean distance of their centers.)

Problem specification

Given are the number of ships N, the limit on moves per turn K and the initial sectors of all ships. Compute which of the two players will win the game if both players play optimally.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case starts with a line containing the number of ships N and the limit K. The i-th of the next N lines contains three integers: the coordinates of the sector where the i-th ship starts.

In the easy input you may assume that the coordinates are small and that in each test case either $K = 1$ or $K = N$.

Output specification

For each test case output a single line with the string `Player 1` if the first player has a winning strategy and the string `Player 2` otherwise.

Example

input

```
2

3 2
2 1 0
0 0 0
47 42 47

2 1
1 0 0
0 1 0
```

output

```
Player 1
Player 2
```

In the first test case the first player can move both ships #1 and #3 straight to $(0, 0, 0)$*. If he does so, he wins.*

In the second test case the first player must move one of the ships to $(0, 0, 0)$*. The second player then moves the other one and wins.*

Lovely stamps

Little Peter is collecting stamps. Recently, he went shopping with his mother Lucia and guess what: As they were going around the post office, he started to blackmail his mom, as only the little boys can. At the post office, they were selling N different types of one-dollar stamps and M different types of two-dollar stamps.

Peter got exactly K dollars from his mom, and he wants to spend all of them on stamps. Note that he can buy more stamps of the same type. You may assume that the post office has an infinite stock of each type of stamp.

Now, Peter is wondering in how many ways he can buy the stamps.

Problem specification

Given are the integers N, M, K, and a prime number P.

Your task is to compute the value $Z \bmod P$, where Z is the (possibly huge) number of ways in which Peter can spend all K dollars on stamps.

Input specification

The first line of the input file contains an integer T specifying the number of test cases. Each test case is preceded by a blank line.

Each test case consists of a single line containing the integers N, M, K, and P.
You may assume that $3 \leq P \leq 1\,000\,000$.
For the small input set, you may assume that $0 \leq N, M \leq 1000$ and $1 \leq K \leq 1000$.
For the large input set, you may assume $0 \leq N, M \leq 300$ and $1 \leq K \leq 1\,000\,000\,000\,000 = 10^{12}$.

Output specification

For each test case output a single line with a single integer: The number of different ways to buy stamps, modulo P.

Example

input

```
3

0 10 2 47

2 2 4 47

5 5 10 47
```

output

```
10
14
6
```

In the first test case, we must buy one 2-dollar stamp and there are 10 types.

In the second test case, we have these options:
– buy two 2-dollar stamps: 3 ways to do so
– buy a 2-dollar stamp and two 1-dollar stamps: $2 \times 3 = 6$ *ways to do so*
– buy four 1-dollar stamps: 5 ways to do so
Therefore the answer is $(3 + 6 + 5) \bmod 47 = 14 \bmod 47 = 14$.

Mass psychoanalysis

Note. This is a special problem. No points can be gained for this problem.
The only thing you can gain is negative penalty time.

In Asimov's famous series of books on the Foundation one of the main characters, Hari Seldon, is credited with inventing psychohistory: a scientific discipline that can use statistics to predict the behavior of large groups of people. We are now asking you to lay the foundations of psychohistory for real. Your goal is to predict the behavior of your fellow contestants as good as you can.

We will be playing a simple game. In the game you have a single submit. You can use this submit at any point during the game. If you do so, we will record the exact **time** between the start of the game and the moment of your submission. During the game these submits are **not shown** in the standings.

Once the game is over, we will then compute the **average** of all the submission times. Your goal in this game: your time must be as close as possible to **two thirds of that average**.

Example

Three teams take part in the game. The first team submits 500 seconds after the game starts, the second team submits at 600 seconds and the third one at 1690 seconds.

The average submission time is $(500 + 600 + 1690)/3 = 2790/3 = 930$ seconds. Two thirds of that average is $930 \cdot (2/3) = 620$. The second team's submission time is closest to 620.

Input specification

You may submit anything you wish to (e.g., an empty file). The content of your submission does not matter, only the time is important.

The **first game** starts at the beginning of the contest and ends after two hours. During these two hours you may submit the **easy data set** M1 in order to play this game.

After two hours the first round will be evaluated and the results **will be made public**.

The **second game** starts at 2h30 and ends at 4h30 into the contest. (I.e., it starts in the middle of the contest and ends 30 minutes before its end.) During these two hours you may submit the **hard data set** M2 in order to play this game. (Note that the time you choose is the time since the beginning of the **game**, not the beginning of the **contest**.)

After the second game ends, it will be evaluated and the results will be made public.

Evaluation

In the first game the following scoring is used. For each team only the best option applies.

- Any team within 5 seconds of the goal gains -120 penalty minutes.
- Any team within 30 seconds of the goal gains -80 penalty minutes.
- Any team within a minute of the goal gains -60 penalty minutes.
- Any team within two minutes of the goal gains -40 penalty minutes.
- Any team within five minutes of the goal gains -20 penalty minutes.

In the second game the scoring will be double (i.e., -240 for an exact guess, etc.).