Problem A: Burger

inputfile: A.IN
outputfile: A.OUT

Description

When Mr. and Mrs. Clinton's twin sons Ben and Bill had their tenth birthday, the party was held at the McDonald's restaurant at South Broadway 202, New York. There were 20 kids at the party, including Ben and Bill. Ronald McDonald had made 10 hamburgers and 10 cheeseburgers and when he served the kids he started with the girl directly sitting left of Bill. Ben was sitting to the right of Bill. Ronald flipped a (fair) coin to decide if the girl should have a hamburger or a cheeseburger, head for hamburger, tail for cheeseburger. He repeated this procedure with all the other 17 kids before serving Ben and Bill last. Though, when coming to Ben he didn't have to flip the coin anymore because there were no cheeseburgers left, only 2 hamburgers.

Problem

Ronald McDonald was quite surprised this happened, so he would like to know what the probability is of this kind of events. Calculate the probability that Ben and Bill will get the same type of burger using the procedure described above. Ronald McDonald always grills the same number of hamburgers and cheeseburgers.

Input

The first line of the input-file contains the number of problems *n*, followed by *n* times:
a line with an even number *[2,4,6..1000]*, which indicates the number of guests present at the party including Ben and Bill.

Output

The output consists of *n* lines with on each line the probability (4 decimals precise) that Ben and Bill get the same type of burger.
Note: a variance of ± 0.0001 is allowed in the output due to rounding differences.

Example

Sample input

```
3
6
10
256
```

Correct output for the sample input

```
0.6250
0.7266
0.9500
```

Problem B: Wormholes

inputfile: B.IN
outputfile: B.OUT

Description

In the year 2163, *wormholes* were discovered. A wormhole is a subspace tunnel through space and time connecting two star systems. Wormholes have a few peculiar properties:

Wormholes are *one-way* only.
The time it takes to travel through a wormhole is negligible.
A wormhole has two end points, each situated in a star system.
A star system may have more than one wormhole end point within its boundaries.
For some unknown reason, starting from our solar system, it is always possible to end up in any star system by following a sequence of wormholes (maybe Earth *is* the centre of the universe).
Between any pair of star systems, there is at most one wormhole in either direction.
There are no wormholes with both end points in the same star system.

All wormholes have a constant time difference between their end points. For example, a specific wormhole may cause the person travelling through it to end up 15 years in the future. Another wormhole may cause the person to end up 42 years in the past.

A brilliant physicist, living on earth, wants to use wormholes to study the Big Bang. Since warp drive has not been invented yet, it is not possible for her to travel from one star system to another one directly. This *can* be done using wormholes, of course.

Problem

The scientist wants to reach a cycle of wormholes somewhere in the universe that causes her to end up in the past. By travelling along this cycle a lot of times, the scientist is able to go back as far in time as necessary to reach the beginning of the universe and see the Big Bang with her own eyes. Write a program to find out whether such a cycle exists.

Input

The input file starts with a line containing the number of cases c to be analysed. Each case starts with a line with two numbers n and m. These indicate the number of star systems $(1 \le n \le 1000)$ and the number of wormholes $(0 \le m \le 2000)$. The star systems are numbered from 0 (our solar system) through $n-1$. For each wormhole a line containing three integer numbers x, y and t is given. These numbers indicate that this wormhole allows someone to travel from the star system numbered x to the star system numbered y, thereby ending up t $(-1000 \le t \le 1000)$ years in the future.

Output

The output consists of c lines, one line for each case, containing the word **`possible`** if it is indeed possible to go back in time indefinitely, or **`not possible`** if this is not possible with the given set of star systems and wormholes.

Example

Sample input

```
2
3 3
0 1 1000
1 2 15
2 1 -42
4 4
0 1 10
1 2 20
2 3 30
3 0 -60
```

Correct output for the sample input

```
possible
not possible
```

Problem C: Squares

inputfile: C.IN
outputfile: C.OUT

Description

The game of *Squares* is not well known in our region, but there are some fanatics who spend their whole day playing Squares, using boards as large as they can find. However, recently the *International Squares Federation* decided that the sides of a Squares board may not be larger than *1000*.

Squares is played by two players on a rectangular board, similar to a very big chess board. The fields are numbered from *(1,1)* at the bottom left corner, to *(1,w)* at the bottom right corner and *(h,w)* at the top right corner, where *h* and *w* are the height and width of the board. Usually, one of the players decides the size of the board, and the other player makes the first move.

A move consists of choosing a free field. This field is extended to the right and to the top of the board to form a square as large as possible without intersecting already occupied fields. The fields of this new square then belong to the player who made the move, and are no longer free. The game is finished when there are no free fields left. However, it is possible for the players to decide to end the game earlier.

The scoring rules are too complicated to explain here. The number of occupied fields plays a role, but also the moment in the game at which those fields were occupied by the player.

For beginning players, the best strategy is to try to occupy as many fields as possible in each move. More experienced players will sometimes occupy smaller squares for strategic reasons.

Problem

Write a program to support beginning players, which, given the size of the board and the moves already made, indicates a field that must be chosen in the next move to occupy a square that is as large as possible.

Input

The input starts with a line which indicates the number of games *g* to be analysed.

For each game, the input begins with a line with three numbers *h*, *w*, and *m*, separated by a space, where *h* and *w* are the height and width of the board $(1 \le h \le 1000, 1 \le w \le 1000)$, and *m* is the number of moves already made $(0 \le m \le 100)$.

For each move, the input file contains one line with the row and column $(1 \le r \le h, 1 \le c \le w)$ of the field the player has chosen, separated by a space. The move is legal according to the rules given above, i.e., the field is free.

Output

The output consists of *g* lines, one line for each game. If there is no legal move possible, the text **`game over`** should be given. Otherwise, the line should contain three numbers *r*, *c*, and *s*, separated by one space, where *r* and *c* are the row and column where a maximal square with side *s* can be formed. If there is more than one possible solution, the one with the smallest *r* is given, and if that is not decisive, the one with the smallest *c*.

Example

Sample input

```
2
8 8 4
8 1
3 6
1 4
2 1
500 1000 2
1 1
1 501
```

Correct output for the sample input

```
5 2 4
game over
```

Problem D: Magic

inputfile: D.IN
outputfile: D.OUT

Description

A well known big computer factory is involved in a very secret project. The project is so secret that we cannot tell you the name of the manufacturer, but we can tell you something about the project. It is called: the Non-Deterministic Magic Holistic New Age Computer (NDMHNAC). It is a major breakthrough in thinking about computing, in that it steps away from every traditional idea in Computer Science and Hardware Design. One of the (many) abilities of NDMHNAC is to execute calculations with very large numbers very fast.

Unfortunately it has a small flaw. Its circuits are based on Magic, so its calculations are disturbed by magic in the same way as traditional circuits are disturbed by electromagnetic fields. As you know the numbers 3 and 7 have strong magic. Therefore these numbers should be avoided, because they will disturb, and eventually destroy the circuits of the machine. The algorithms used internally in the machine never generate offending numbers. The only problem is to ban offending numbers from input.

To that purpose a conventional machine is used as a pre-processor. This machine should transform offending numbers to admissible numbers. (You might wonder how a computer can give a correct answer if you jumble its input. Now this question shows how much you are tied to traditional thinking in computer science. A Non-Deterministic Magic Holistic New Age Computer will give correct answers, even if the input is wrong! (Compare this to traditional computers giving wrong answers, even if the input is correct!)).

Numbers are arbitrarily large, non-negative integers, in decimal representation. A number is offending if it has one of the following properties:
it is divisible by 3, or by 7 (e.g. 6, 14, 21),
a 3 or 7 occurs in its decimal representation, (e.g. 13, 27, 37),
a digit is repeated 3 or 7 times consecutively in its decimal representation. (e.g. 2411145, 10000000).
(NB the number 0 turns out not to be offending).

Offending numbers are to be transformed into non-offending numbers, using (repeatedly) the following rules:
if a number is divisible by 3: divide it by 3
if a number is divisible by 7: divide it by 7
if a 3 occurs in the decimal representation: remove this 3 (remove if possible leading zeros)
if a 7 occurs in the decimal representation: remove this 7 (remove if possible leading zeros)
if a substring of 3 times the same digit occurs in the decimal representation: remove it (remove if possible leading zeros)
if a substring of 7 times the same digit occurs in the decimal representation: remove it (remove if possible leading zeros)
if eventually no digit is left, consider this as the number 0.

A traditional programmer is hired to implement these rules. After reading the specification given above, he complains that it is highly ambiguous. For example, the third rule is about removing a 3. Should all occurrences of 3 be removed, or just any, or only the first one? Moreover, should every rule be applied repeatedly, or should all rules be applied, and then all rules again, and in which order? Different interpretations of the specification may give rise to thousands of different answers.

The designers of the new machine declare that they do not care, as long as a non-offending number is obtained. Seeing that the programmer feels really uncomfortable, they say: 'OK, just give us the largest possible answer'. The programmer got himself an other job.

Problem

Write a program which, given a number, generates the largest non-offending number that can be obtained by applying the rules given above.

Input

The first line of the input-file contains the number of problems *n*. The next *n* lines contain one number each. An input line is never longer than *21* characters ($21 = 3 \times 7$). An input line never starts with *0*.

Output

The outputfile consists of *n* lines with on each line the largest non-offending number that can be obtained by applying the rules given above.

Example

Sample input

```
3
999
273
2331
```

Correct output for the sample input

```
11
2
11
```

Problem E: Jackpot

inputfile: E.IN
outputfile: E.OUT

Description

An addicted gambler walks from his home to the casino every day to try his luck.
He likes most games but the one he likes most is the slot machine, also known as the jackpot or the one-armed-bandit. Because of his years of experience he exactly knows which, and how many, symbols appear on the wheels of every slot machine.

Unfortunately all wheels have changed since last time he was in the casino. The wheels are completely new, vary in size and are much larger than the former ones. He wonders if there's a way to find out which slot machine would give him the most profit. He suspects that a lot of calculations are involved so he decides that this job can better be done by a computer.

Problem

Can you build a program for this man which calculates the pay-rate of a number of machines, if the complete wheels are known? There are three wheels on each machine, all three wheels rotate independently of each other. Each line of three times the same symbol pays an amount of money. The display of the machine consists of three visible rows. The middle row pays 10 coins if three identical symbols are shown, both the upper and the lower row pay 5 coins. In addition, the line from the upper left corner to the lower right as well as the line from the lower left to the upper right corner can result in a winning combination. They both pay 7 coins. The pay-rate is defined as the average (or expected) amount of money paid by the machine on each play.

Input

The first line gives the number of slot machines s. Then for each slot machine follows:
a line with the numbers x, y, z, each separated by at least one space $(3 \leq x,y,z \leq 200)$.
x is the number of symbols on the first wheel, y the number of symbols on the second wheel and z the number of symbols on the last wheel.
3 lines with the symbols on each wheel. A symbol is represented by one capital character. The first line contains the symbols of the first wheel, the second line contains the symbols of the second wheel and the third line contains the symbols of the third wheel.

Output

The output consists of s lines with on each line the pay-rate of the corresponding slot machine, with exact 4 digits after the decimal point.
Note: a variance of ± 0.0001 is allowed in the output due to rounding differences.

Example

Sample input

```
2
3 4 6
AAB
BABA
BBAAAB
12 15 18
CCCCCCCCCCCC
CCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCC
```

Correct output for the sample input

```
8.5000
34.0000
```

Problem F: Dividing coins

inputfile: F.IN
outputfile: F.OUT

Description

It's commonly known that the Dutch have invented copper-wire. Two Dutch men were fighting over a nickel, which was made of copper. They were both so eager to get it and the fighting was so fierce, they stretched the coin to great length and thus created copper-wire.

Not commonly known is that the fighting started, after the two Dutch tried to divide a bag with coins between the two of them. The contents of the bag appeared not to be equally divisible. The Dutch of the past couldn't stand the fact that a division should favour one of them and they always wanted a fair share to the very last cent. Nowadays fighting over a single cent will not be seen anymore, but being capable of making an equal division as fair as possible is something that will remain important forever…

That's what this whole problem is about. Not everyone is capable of seeing instantly what's the most fair division of a bag of coins between two persons. Your help is asked to solve this problem.

Problem

Given a bag with a maximum of 100 coins, determine the most fair division between two persons. This means that the difference between the amount each person obtains should be minimised. The value of a coin varies from 1 cent to 500 cents. It's not allowed to split a single coin.

Input

A line with the number of problems n, followed by n times:
a line with a non negative integer m $(m \leq 100)$ indicating the number of coins in the bag
a line with m numbers separated by one space, each number indicates the value of a coin.

Output

The output consists of n lines. Each line contains the minimal positive difference between the amount the two persons obtain when they divide the coins from the corresponding bag.

Example

Sample input

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

Correct output for the sample input

```
0
1
```

Problem G: Crimewave

inputfile: G.IN
outputfile: G.OUT

Description

Nieuw Knollendam is a very modern town. This becomes clear already when looking at the layout of its map, which is just a rectangular grid of streets and avenues. Being an important trade centre, Nieuw Knollendam also has a lot of banks. Almost on every crossing a bank is found (although there are never two banks at the same crossing). Unfortunately this has attracted a lot of criminals. Bank hold-ups are quite common, and often on one day several banks are robbed. This has grown into a problem, not only to the banks, but to the criminals as well. After robbing a bank the robber tries to leave the town as soon as possible, most of the times chased at high speed by the police. Sometimes two running criminals pass the same crossing, causing several risks: collisions, crowds of police at one place and a larger risk to be caught.

To prevent these unpleasant situations the robbers agreed to consult together. Every Saturday night they meet and make a schedule for the week to come: who is going to rob which bank on which day? For every day they try to plan the get-away routes, such that no two routes use the same crossing. Sometimes they do not succeed in planning the routes according to this condition, although they believe that such a planning should exist.

Problem

Given a grid of ($s \times a$) and the crossings where the banks to be robbed are located, find out whether or not it is possible to plan a get-away route from every robbed bank to the city-bounds, without using a crossing more than once.

Input

The first line of the input contains the number of problems p to be solved.
The first line of every problem contains the number s of streets $(1 \leq s \leq 50)$, followed by the number a of avenues $(1 \leq a \leq 50)$, followed by the number b $(b \geq 1)$ of banks to be robbed.
Then b lines follow, each containing the location of a bank in the form of two numbers x (the number of the street) and y (the number of the avenue). Evidently $1 \leq x \leq s$ and $1 \leq y \leq a$.

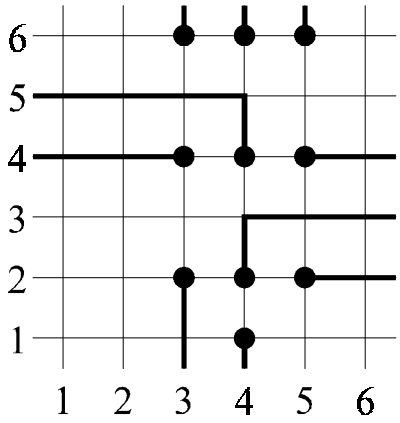
Output

The output file consists of p lines. Each line contains the text **`possible`** or **`not possible`**. If it is possible to plan non-crossing get-away routes, this line should contain the word: **`possible`**. If this is not possible, the line should contain the word **`not possible`**.

Example

Sample input

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



Correct output for the sample input

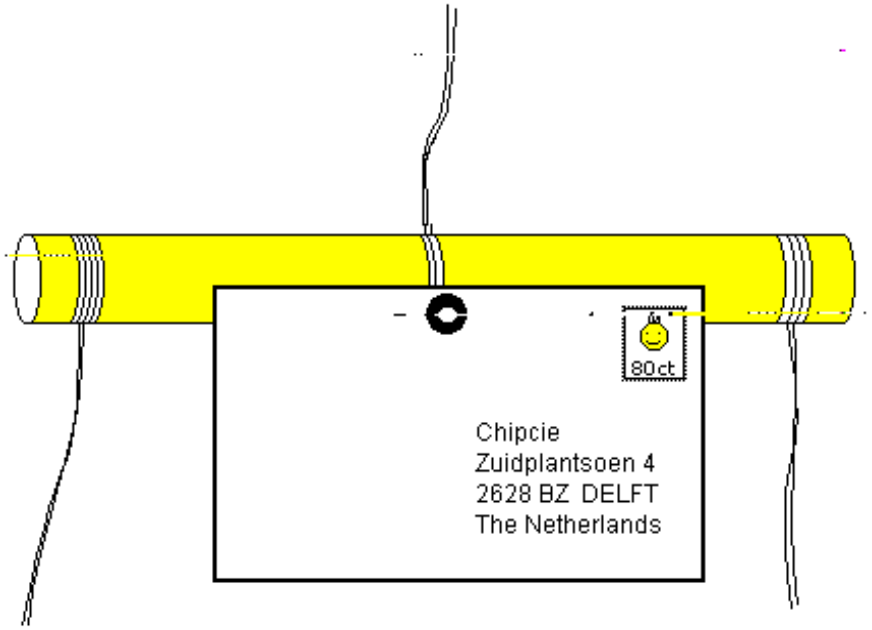
```
possible
not possible
```

Problem H: Gaston

inputfile: H.IN
outputfile: H.OUT

Description

Gaston is an employee of a famous comic magazine. One of his tasks (the one he dislikes most) is archiving post. He has some delay in archiving, so the larger part of his desk is occupied by an enormous pile of unprocessed letters. Sometimes one of the editors needs a certain letter. Then Gaston is faced with the problem to find that letter in the pile.



Gaston realised that he has a retrieval problem. Using sticks and threads he designed a system for better retrieving not archived letters. In the middle of a stick he fixes a letter. At both ends of the stick a thread is fixed where another stick (with letter) can be fastened, etc.
He arranges the letters such that all letters pending under the left hand thread of a stick are alphabetically before the letter in the middle. The letters pending under the right hand thread are alphabetically after the letter in the middle. This is true for every stick, so Gaston obtained a sorted tree, in which it is easy to find a letter.

Making a prototype, Gaston finds out that he has a balancing problem. If there is a difference in weight between the ends of a stick, the stick is unbalanced, and the whole system collapses in a mess even bigger than before. Using some extra thread, Gaston invents an asymmetric fix: now the stick is still balanced if the left hand side contains one more letter than the right hand side. If the right hand side contains more letters, the stick is always unbalanced, so the fix is indeed asymmetric.

Opinions about Gaston are different. He believes to be a genius himself, the others don't. The editors point out that when a letter is added there is a fair chance that the tree gets out of balance and has to be rearranged. We want you to find out how often the tree (or a subtree) must be rearranged, given a tree and a sequence of incoming letters.

Problem

We have simplified the problem a bit. All letters have the same weight. Every letter has a unique (positive) number. The letters have to be ordered by increasing number. A stick is balanced if the left hand side contains the same number of letters as the right hand side, or at most one more.

A letter is added according to the following procedure. If its number is lower than the number of the letter on the stick, it is added to the left hand side of the balance. If its number is higher, it is added to the right hand side. Following this procedure repeatedly, eventually a free thread is found. Then the letter is fixed on a fresh stick (having two free threads, one at each end) which is tied to the free thread. Then, starting at the top working downwards, all sticks are checked for being balanced. If a stick is unbalanced, a balancing step is executed. Because a balancing step might result in having other sticks being unbalanced again, the whole tree is checked again from top to bottom for unbalanced sticks. This is done until no unbalanced sticks are found.

A balancing step is defined as bringing one stick *s*, being unbalanced, in a balancing state. This is done by removing one letter on the side of the stick which is too heavy and adding one on the other side. This is done according to the following procedure:
first remove the letter l_{old} at the stick *s* itself
then, find an appropriate letter on the side which is too heavy and fix this one on the stick *s*
finally add the former letter l_{old} on the other side.
A balancing step can result in more sticks being unbalanced! Because the new letter l_{new} at the stick is removed from it's old position, an unbalanced stick can be left behind. Also adding the former letter can result in having a new unbalanced stick. Getting these sticks balanced again is not a part of this balancing step(!), but requires new balancing steps.

The problem is to find out how many balancing steps are needed to keep the tree balanced when new letters are added.

Notation

At the end of a thread we may find:
a stick, with a letter, and a thread at both ends, or
nothing

This leads to the following textual representation of a balance.
if it is a stick: the number of the letter on it, followed by the representation of what is at the end of its left hand thread, followed by the representation of what is at the end of its right hand thread.
if it is a loose thread: the number 0.

Within a line numbers are separated with at least one space. When needed the textual representation of a balance may run over several lines. Every line contains at least one number.

Input

A line with the number *n* of problems, then *n* times:
a description of a balance (see notation)
on a separate line the number *b* of letters to be added, then, beginning on the next line
b times the number of a letter *l (1 ≤ l ≤ 1000)*.

Within a line the numbers of the letters to insert are separated with at least one space. The numbers of the letters may run over several lines. Every line contains at least one number.

Output

The output consists of *n* lines,. Each line contains the number of balancing steps needed to keep the tree balanced when the letters are added.

Example

Sample input

```
2
3 2 0 0
5 0 0
4
1 6 7
9
400 250 0 0 0
2
511 100
```

Correct output for the sample input

```
3
0
```