ACM International Collegiate Programming Contest 1995

Northwestern European Regionals

Problem A

Modern Art

The famous painter Mel Borp is working on a brilliant series of paintings that introduce a new experimental style of Modern Art. At first glance, these paintings look deceptively simple, since they consist only of triangles of different sizes that seem to be stacked on top of each other. Painting these works, however, takes an astonishing amount of consideration, calculation, and precision since all triangles are painted without taking the brush off the canvas. How exactly Mel paints his works is a well-kept secret.

Recently, he started on the first painting of his new series. It was a single triangle, titled $T_{0,0}$. After that, he created $T_{1,1}$, the basis for his other works (see Figure 1).

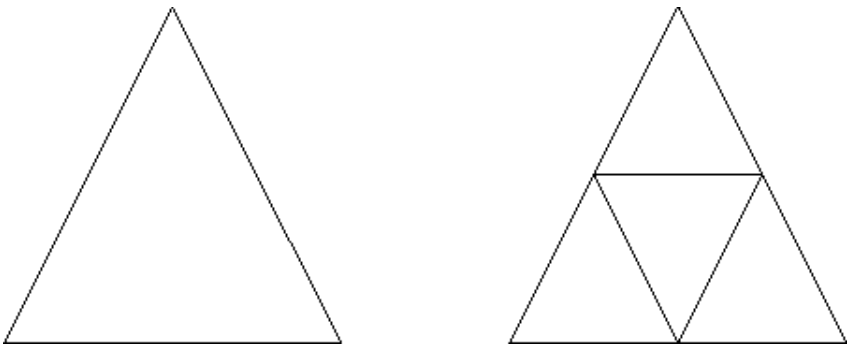


Figure: Early work: $T_{0,0}$ (left) and $T_{1,1}$ (right).

Then he decided to take his experimenting one step further, and he painted $T_{1,2}$ and $T_{3,2}$. Compare Figure 1 and Figure 2 to fully appreciate the remarkable progression in his work.

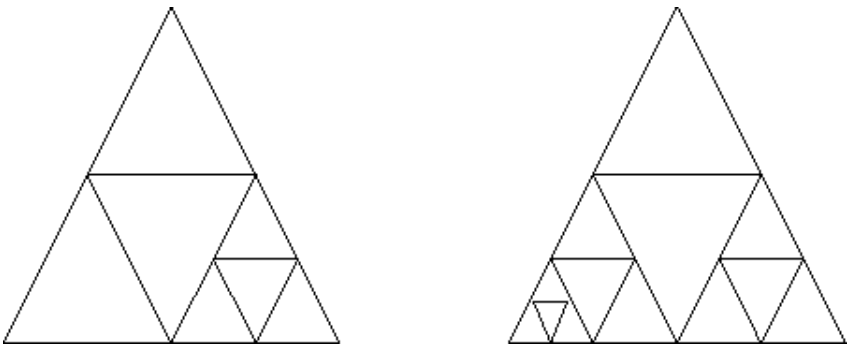


Figure: Advanced work: $T_{1,2}$ (left) and $T_{3,2}$ (right).

Note that the shape of the painting can be deduced from its title, , as follows:

Note that the shape of the painting can be deduced from its title, $T_{p,q}$, as follows:

- $T_{0,0}$ is a single triangle (see Figure 1);
- $T_{1,1}$ consists of a smaller, inverted triangle placed inside $T_{0,0}$, so that the result consists of four smaller triangles (see Figure 1);
- if $p > 0$ and $q > 1$, then the painting looks like $T_{p,q-1}$, except that an inverted triangle has been placed inside the bottom-right triangle of $T_{p,q-1}$, splitting it into four smaller triangles (compare for example $T_{1,1}$ and $T_{1,2}$);
- if $p > 1$ and $q > 0$, the painting looks like $T_{p-1,q}$, except that an inverted triangle has been placed inside the bottom-left triangle of $T_{p-1,q}$, splitting it into four smaller triangles;
- other values of p and q: it is not a valid title of a painting.

The triangles of a painting look all the same (each triangle is an isosceles triangle with two sides of the same length), but their height and width depend on the size of the canvas Mel used.

Mel wanted to end the series with $T_{10,10}$, the most complex painting he thought he would be able to paint. But no matter how many times he tried, he could not get it right. Now he is desperate, and he hopes you can help him by writing a program that prints, in order, the starting and ending coordinates of the lines Mel has to paint. Of course, you will need to know how Mel paints his works, so we will now reveal his secret technique.

As an example, take a look at $T_{1,2}$ (see Figure 3):

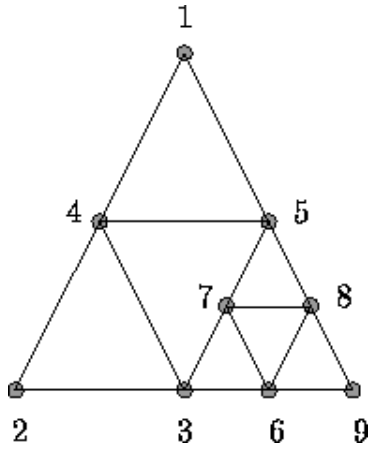


Figure: $T_{1,2}$.

Mel always starts at the top of the top triangle, drawing a line straight to the lower-left corner of the lower left triangle (in this example, 1-2), continuing with a line to the lower-right corner of that triangle (in this example, 2-3). Next, he works his way up by drawing a line to the top of that triangle (in this example, 3-4). Now he has either reached the starting point again (finishing yet another masterpiece) or he has reached the lower-left corner of another triangle (in this example, 1-4-5). In the latter case, he continues by drawing the bottom line of that triangle (4-5) and after that he starts working on the triangle or triangles that is or are located underneath the lower-right corner of that triangle, in the same way. So he continues with (5-3), (3-6), (6-7), (7-8), (8-6), (6-9),

and (9-1) as the finishing touch.

Input Specification

The first line of the input contains the number of test cases. Each test case consists of one line containing four non-negative integers p, q, x, and y, separated by spaces. $T_{p,q}$ is the title of the painting and (x,y) are the coordinates of the top of the top triangle. Further, $p, q \leq 10$ and x,y < 32768. All triangles have a nonzero area.

Output Specification

For every test case, the output contains the pairs of (x,y) integer coordinates of the starting and ending points of all lines Mel has to draw for the painting $T_{p,q}$ in the right order, in the format

$$(startX, startY)(endX, endY)$$

followed by a newline. The output for each test case must be followed by an empty line.

Sample Input

```
2
0 0 1 1
1 2 512 1024
```

Sample Output

```
(1,1)(0,0)
(0,0)(2,0)
(2,0)(1,1)

(512,1024)(0,0)
(0,0)(512,0)
(512,0)(256,512)
(256,512)(768,512)
(768,512)(512,0)
(512,0)(768,0)
(768,0)(640,256)
(640,256)(896,256)
(896,256)(768,0)
(768,0)(1024,0)
(1024,0)(512,1024)
```

ACM International Collegiate Programming Contest 1995

Northwestern European Regionals

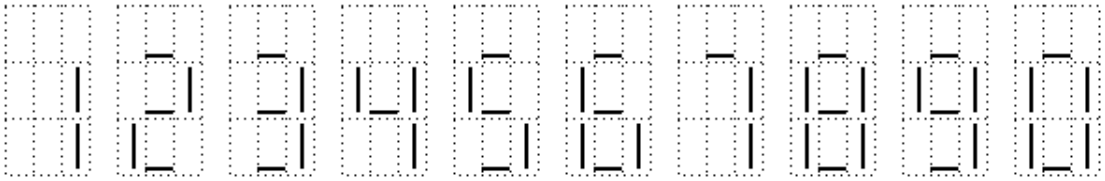
Problem B

Bank (Not Quite O.C.R.)

Banks, always trying to increase their profit, asked their computer experts to come up with a system that can read bank cheques; this would make the processing of cheques cheaper. One of their ideas was to use optical character recognition (ocr) to recognize bank accounts printed using 7 line-segments.

Once a cheque has been scanned, some image processing software would convert the horizontal and vertical bars to ASCII bars '|' and underscores '_-'.

The ASCII 7-segment versions of the ten digits look like this:

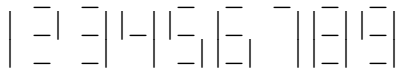


A bank account has a 9-digit account number with a checksum. For a valid account number, the following equation holds: $(d_1+2 * d_2 + 3 * d_3 + ... + 9 * d_9) \bmod 11 = 0$. Digits are numbered from right to left like this: $d_9d_8d_7d_6d_5d_4d_3d_2d_1$.

Unfortunately, the scanner sometimes makes mistakes: some line-segments may be missing. Your task is to write a program that deduces the original number, assuming that:

- when the input represents a valid account number, it is the original number;
- at most one digit is garbled;
- the scanned image contains no extra segments.

For example, the following input



used to be ''123456789''.

Input Specification

The input file starts with a line with one integer specifying the number of account numbers that

have to be processed. Each account number occupies 3 lines of 27 characters.

Output Specification

For each test case, the output contains one line with 9 digits if the correct account number can be determined, the string ``failure'' if no solutions were found and ``ambiguous'' if more than one solution was found.

Sample Input

```
4
    _  _     _  _  _  _  _ 
  | _| _||_||_ |_   ||_||_|
  | _  _|  | _||_|  ||_| _|
    _  _  _  _  _  _     _ 
|_||_|| || ||_   |  |  ||_ 
  | _||_||_||_|  |  |  | _|
 _  _  _  _  _  _  _  _  _ 
|_||_||_||_||_||_||_||_||_|
|_||_||_||_||_||_||_||_||_|
 _     _  _  _  _  _  _  _ 
|_|  ||_||_||_||_||_||_||_|
|_|  ||_||_||_||_||_||_||_|
```

Sample Output

```
123456789
ambiguous
failure
878888888
```

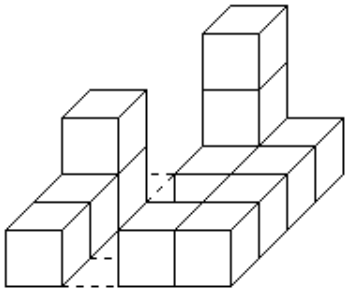
ACM International Collegiate Programming Contest 1995

Northwestern European Regionals

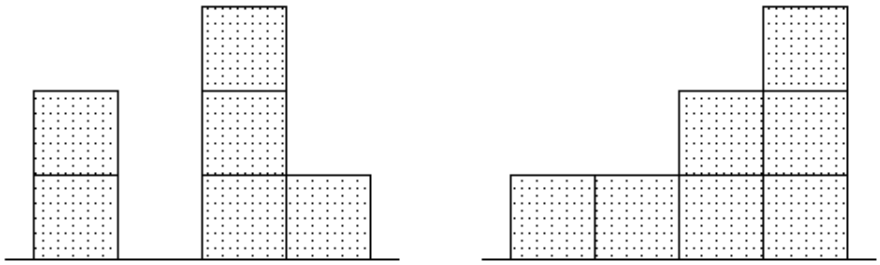
Problem C

Matty's Blocks

Little Matty liked playing with his blocks very much. He always constructed his 'buildings' in the same way: he made stacks of one or more blocks, and he put those stacks on a square table that was exactly K blocks wide ($K = 8$ for his largest table, so it could contain up to 8×8 block stacks). He didn't put the stacks randomly on the table. No, he always made a nice 'square' pattern. In most buildings, there was no pattern visible in the heights of the stacks. However, since Little Matty himself was only eight blocks tall, a single stack of blocks never consisted of more than eight blocks. This is an example of one of his buildings. It was built on a table that could contain 4×4 block stacks.



He liked drawing too. To record his block buildings for future generations, he would draw them on paper. Since drawing a three-dimensional block building just was too hard for him, he made two drawings of a building: one straight from the front (you could only see the front of the blocks), and one from the right (you could only see the right side of the blocks). The drawings were in fact two-dimensional projections of the block building, showing only its outline on the front or on the right side. These are the drawings he made of the building shown above.

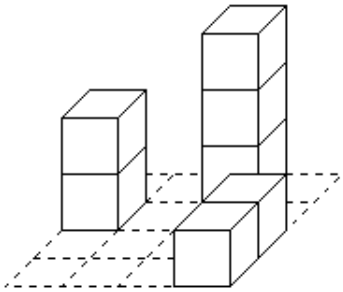


Front Right side

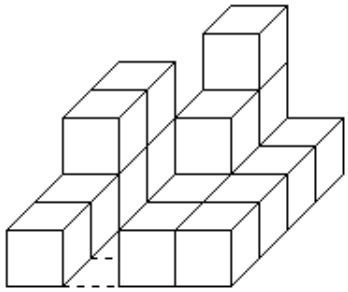
He thought that such a pair of drawings would give enough information to be able to re-build the block building later (but he never tried it).

Years later, looking again at the drawings, he realized that this was not the case: from most pairs of drawings, he was able to construct more than one building that had the same outlines (front and right side) as the original one. He found out that from every (front, side)-pair of drawings that he had made, he could construct a 'minimal' building, one that has the same outlines as the original building and contains a minimal number of blocks N (so it was not possible to construct a building with the same outlines with less than N blocks). Furthermore, he could add more blocks to this minimal building, so that the outlines remained the same, up to the point that he had added M () blocks and he had a 'maximal' building, one that still had the same outlines as his minimal one, and adding one extra block would result in a building with a different outline (so there are no buildings with the same outlines that contain more than $N + M$ blocks).

As an example, these are minimal and maximal buildings for the drawings shown above. In this case, $N = 7$ and $M = 10$.



Minimal Building



Maximal Building

Matty started determining the N and M for every pair of drawings he had made, but soon he found this task to be tedious. Now he asks *you* to write a program that does the job for him!

Input Specification

The input contains on the first line the number of test cases. Each test case starts with a line containing only the size of the table K. The next pair of lines each contain the description of one drawing. Each description consists of K non-negative integers separated by spaces. Each number indicates the height of the corresponding projection of a stack of blocks in the drawing. The description of the front drawing always precedes the description of the right side drawing. From each pair of drawings at least one block building can be constructed.

Output specification

For each test case output the following line:

`Matty needs at least N blocks, and can add at most M extra blocks.`

Sample Input

```
2
4
2 0 3 1
1 1 2 3
1
```

```
1
1
```

Sample Output

```
Matty needs at least 7 blocks, and can add at most 10 extra blocks.
Matty needs at least 1 blocks, and can add at most 0 extra blocks.
```

ACM International Collegiate Programming Contest 1995

Northwestern European Regionals

Problem D

Block Voting

Different types of electoral systems exist. In a block voting system the members of a party do not vote individually as they like, but instead they must collectively accept or reject a proposal. Although a party with many votes clearly has more power than a party with few votes, the votes of a small party can nevertheless be crucial when they are needed to obtain a majority. Consider for example the following five-party system:

party	votes
A	7
B	4
C	2
D	6
E	6

Coalition {A,B} has 7 + 4 = 11 votes, which is not a majority. When party C joins coalition {A,B}, however, {A,B,C} becomes a winning coalition with 7+4+2 = 13 votes. So even though C is a small party, it can play an important role.

As a measure of a party's power in a block voting system, John F. Banzhaf III proposed to use the *power index*. 1 The key idea is that a party's power is determined by the number of minority coalitions that it can join and turn into a (winning) majority coalition. Note that the empty coalition is also a minority coalition and that a coalition only forms a majority when it has more than half of the total number of votes. In the example just given, a majority coalition must have at least 13 votes.

In an ideal system, a party's power index is proportional to the number of members of that party.

Your task is to write a program that, given an input as shown above, computes for each party its power index.

Input Specification

The first line contains a single integer which equals the number of test cases that follow. Each of the following lines contains one test case.

The first number on a line contains an integer P in $[1 \ldots 20]$ which equals the number of parties for that test case. This integer is followed by P positive integers, separated by spaces. Each of these integers represents the number of members of a party in the electoral system. The i-th number represents party number i. No electoral system has more than 1000 votes.

Output Specification

For each test case, you must generate *P* lines of output, followed by one empty line. *P* is the number of parties for the test case in question. The *i*-th line (*i* in [1...*P*]) contains the sentence:

```
party i has power index I
```

where *I* is the power index of party *i*.

Sample Input

```
3
5 7 4 2 6 6
6 12 9 7 3 1 1
3 2 1 1
```

Sample Output

```
party 1 has power index 10
party 2 has power index 2
party 3 has power index 2
party 4 has power index 6
party 5 has power index 6
party 1 has power index 18
party 2 has power index 14
party 3 has power index 14
party 4 has power index 2
party 5 has power index 2
party 6 has power index 2
party 1 has power index 3
party 2 has power index 1
party 3 has power index 1
```