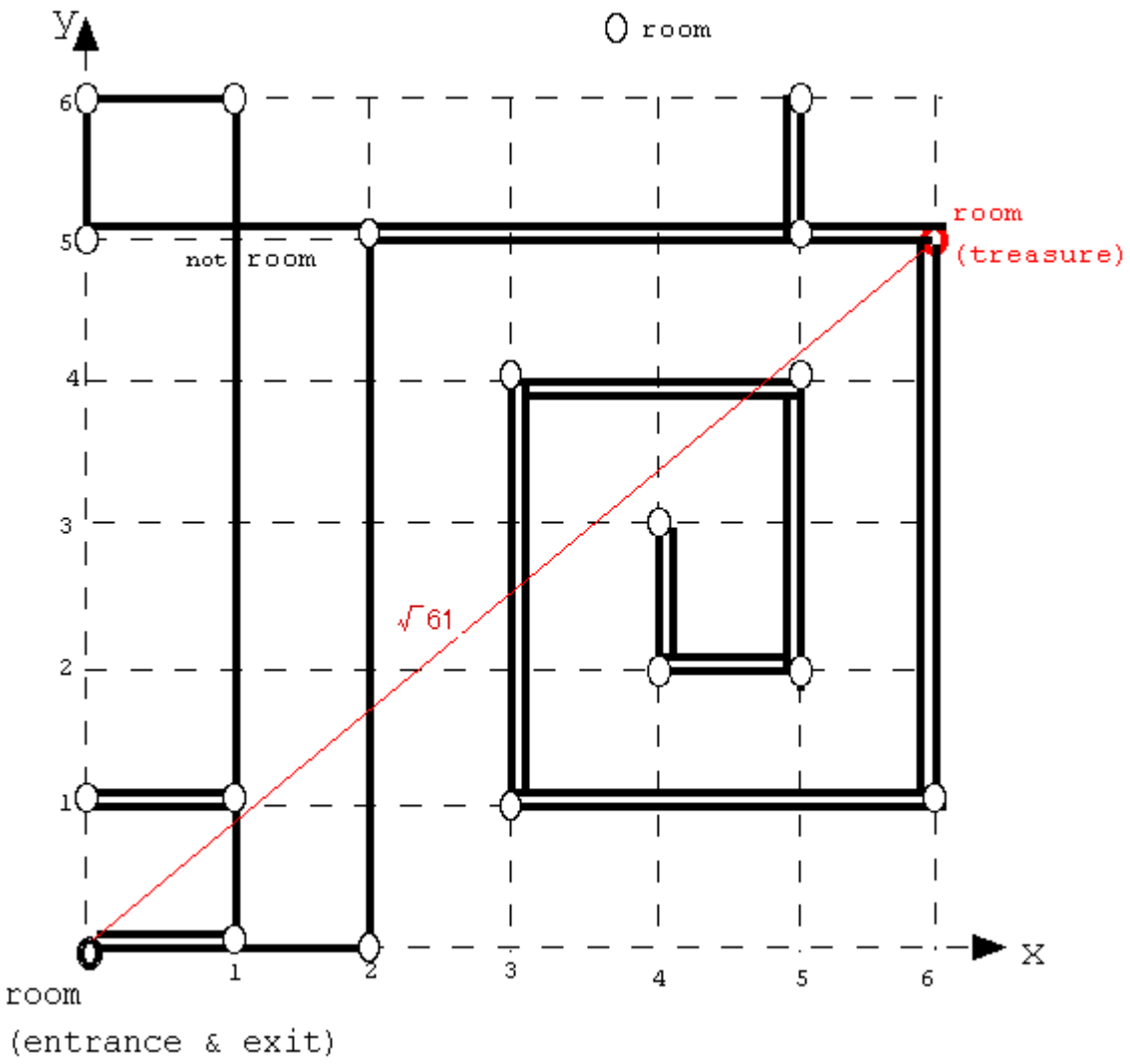
Problem A
Exploring Caves

There are many caves deep in mountains found in the countryside. In legend, each cave has a treasure hidden within the farthest room from the cave's entrance. The Shogun has ordered his Samurais to explore these caves with Karakuri dolls (robots) and to find all treasures. These robots move in the caves and log relative distances and directions between rooms.

Each room in a cave is mapped to a point on an integer grid (x, y >= 0). For each cave, the robot always starts from its entrance, runs along the grid and returns to the entrance (which also serves as the exit). The treasure in each cave is hidden in the farthest room from the entrance, using Euclid distance for measurement, i.e. the distance of the room at point (x, y) from the entrance (0, 0) is defined as the square root of (x*x+y*y). If more than one room has the same farthest distance from the entrance, the treasure is hidden in the room having the greatest value of x. While the robot explores a cave, it records how it has moved. Each time it takes a new direction at a room, it notes the difference (dx, dy) from the last time it changed its direction. For example, suppose the robot is currently in the room at point (2, 4). If it moves to room (6, 4), the robot records (4, 0), i.e. dx=6−2 and dy=4−4. The first data is defined as the difference from the entrance. The following figure shows rooms in the first cave of the Sample Input. In the figure, the farthest room is the square root of 61 distant from the entrance.



Based on the records that the robots bring back, your job is to determine the rooms where treasures are hidden.

Input

In the first line of the input, an integer N showing the number of caves in the input is given. Integers dx_i and dy_i are i-th data showing the differences between rooms. Note that since the robot moves along the grid, either dx_i or dy_i is zero. An integer pair $dx_i = dy_i = 0$ signals the end of one cave's data which means the robot finished the exploration for the cave and returned back to the entrance. The coordinates are limited to (0,0)–(1000,1000).

Output

Print the position (x, y) of the treasure room on a line for each cave.

Sample Input

```
3
1 0
0 1
-1 0
1 0
0 5
-1 0
0 -1
5 0
0 1
0 -1
1 0
0 -4
-3 0
0 3
2 0
0 -2
-1 0
0 1
0 -1
1 0
0 2
-2 0
0 -3
3 0
0 4
-4 0
0 -5
-2 0
0 0
1 0
-1 0
0 0
2 0
0 1
-1 0
0 1
-1 0
0 -2
0 0
```

Output for the Sample Input

```
6 5
1 0
2 1
```

Problem B
Hanafuda Shuffle

There are a number of ways to shuffle a deck of cards. Hanafuda shuffling for Japanese card game 'Hanafuda' is one such example. The following is how to perform Hanafuda shuffling.

There is a deck of *n* cards. Starting from the *p*–th card from the top of the deck, *c* cards are pulled out and put on the top of the deck, as shown in Figure 1. This operation, called a cutting operation, is repeated.

Write a program that simulates Hanafuda shuffling and answers which card will be finally placed on the top of the deck.

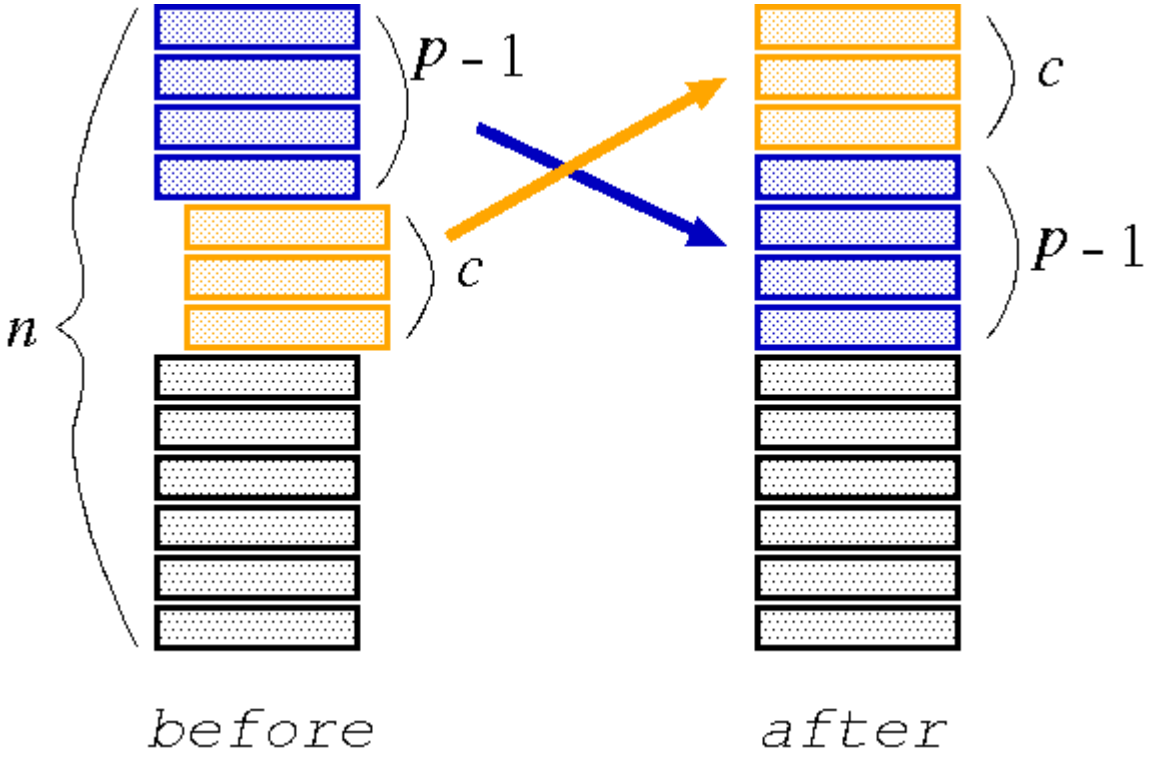


Figure 1: Cutting operation

Input

The input consists of multiple data sets. Each data set starts with a line containing two positive integers *n* (1 <= *n* <= 50) and *r* (1 <= *r* <= 50); *n* and *r* are the number of cards in the deck and the number of cutting operations, respectively.

There are *r* more lines in the data set, each of which represents a cutting operation. These cutting operations are performed in the listed order. Each line contains two positive integers *p* and *c* (*p* + *c* <= *n* + 1). Starting from the *p*–th card from the top of the deck, *c* cards should be pulled out and put on the top.

The end of the input is indicated by a line which contains two zeros.

Each input line contains exactly two integers separated by a space character. There are no other characters in the line.

Output

For each data set in the input, your program should write the number of the top card after the shuffle.

Assume that at the beginning the cards are numbered from 1 to *n*, from the bottom to the top. Each number should be written in a separate line without any superfluous characters such as leading or following spaces.

Sample Input

```
5 2
3 1
3 1
10 3
1 10
10 1
8 3
0 0
```

Output for the Sample Input

```
4
4
```

Problem C
Die Game

Life is not easy. Sometimes it is beyond your control. Now, as contestants of ACM ICPC, you might be just tasting the bitter of life. But don't worry! Do not look only on the dark side of life, but look also on the bright side. Life may be an enjoyable game of chance, like throwing dice. Do or die! Then, at last, you might be able to find the route to victory.

This problem comes from a game using a die. By the way, do you know a die? It has nothing to do with "death." A die is a cubic object with six faces, each of which represents a different number from one to six and is marked with the corresponding number of spots. Since it is usually used in pair, "a die" is a rarely used word. You might have heard a famous phrase "the die is cast," though.

When a game starts, a die stands still on a flat table. During the game, the die is tumbled in all directions by the dealer. You will win the game if you can predict the number seen on the top face at the time when the die stops tumbling.

Now you are requested to write a program that simulates the rolling of a die. For simplicity, we assume that the die neither slips nor jumps but just rolls on the table in four directions, that is, north, east, south, and west. At the beginning of every game, the dealer puts the die at the center of the table and adjusts its direction so that the numbers one, two, and three are seen on the top, north, and west faces, respectively. For the other three faces, we do not explicitly specify anything but tell you the golden rule: the sum of the numbers on any pair of opposite faces is always seven.

Your program should accept a sequence of commands, each of which is either "north", "east", "south", or "west". A "north" command tumbles the die down to north, that is, the top face becomes the new north, the north becomes the new bottom, and so on. More precisely, the die is rotated around its north bottom edge to the north direction and the rotation angle is 90 degrees. Other commands also tumble the die accordingly to their own directions. Your program should calculate the number finally shown on the top after performing the commands in the sequence. Note that the table is sufficiently large and the die never falls off during the game.

Input

The input consists of one or more command sequences, each of which corresponds to a single game. The first line of a command sequence contains a positive integer, representing the number of the following command lines in the sequence. You may assume that this number is less than or equal to 1024. A line containing a zero indicates the end of the input. Each command line includes a command that is one of `north`, `east`, `south`, and `west`. You may assume that no white space occurs in any lines.

Output

For each command sequence, output one line containing solely the number on the top face at the time when the game is finished.

Sample Input

```
1
north
3
north
east
south
0
```

Output for the Sample Input

```
5
1
```

Problem D
Numeral System

Prof. Hachioji has devised a new numeral system of integral numbers with four lowercase letters "m", "c", "x", "i" and with eight digits "2", "3", "4", "5", "6", "7", "8", "9". He doesn't use digit "0" nor digit "1" in this system.

The letters "m", "c", "x" and "i" correspond to 1000, 100, 10 and 1, respectively, and the digits "2", ...,"9" correspond to 2, ..., 9, respectively. This system has nothing to do with the Roman numeral system.

For example, character strings

"5m2c3x4i", "m2c4i" and "5m2c3x"

correspond to the integral numbers 5234 (=5*1000+2*100+3*10+4*1), 1204 (=1000+2*100+4*1), and 5230 (=5*1000+2*100+3*10), respectively. The parts of strings in the above example, "5m", "2c", "3x" and "4i" represent 5000 (=5*1000), 200 (=2*100), 30 (=3*10) and 4 (=4*1), respectively.

Each of the letters "m", "c", "x" and "i" may be prefixed by one of the digits "2", "3", ..., "9". In that case, the prefix digit and the letter are regarded as a pair. A pair that consists of a prefix digit and a letter corresponds to an integer that is equal to the original value of the letter multiplied by the value of the prefix digit.

For each letter "m", "c", "x" and "i", the number of its occurrence in a string is at most one. When it has a prefix digit, it should appear together with the prefix digit. The letters "m", "c", "x" and "i" must appear in this order, from left to right. Moreover, when a digit exists in a string, it should appear as the prefix digit of the following letter. Each letter may be omitted in a string, but the whole string must not be empty. A string made in this manner is called an *MCXI-string*.

An MCXI-string corresponds to a positive integer that is the sum of the values of the letters and those of the pairs contained in it as mentioned above. The positive integer corresponding to an MCXI-string is called its MCXI-value. Moreover, given an integer from 1 to 9999, there is a unique MCXI-string whose MCXI-value is equal to the given integer. For example, the MCXI-value of an MCXI-string "m2c4i" is 1204 that is equal to 1000 + 2*100 + 4*1. There are no MCXI-strings but "m2c4i" that correspond to 1204. Note that strings "1m2c4i", "mcc4i", "m2c0x4i", and "2cm4i" are not valid MCXI-strings. The reasons are use of "1", multiple occurrences of "c", use of "0", and the wrong order of "c" and "m", respectively.

Your job is to write a program for Prof. Hachioji that reads two MCXI-strings, computes the sum of their MCXI-values, and prints the MCXI-string corresponding to the result.

Input

The input is as follows. The first line contains a positive integer n (<= 500) that indicates the number of the following lines. The k+1 th line is the specification of the k th computation (k=1, ..., n).

n
$specification_1$
$specification_2$
...
$specification_n$

Each specification is described in a line:

MCXI-string$_1$ *MCXI-string*$_2$

The two MCXI-strings are separated by a space.

You may assume that the sum of the two MCXI-values of the two MCXI-strings in each specification is less than or equal to 9999.

Output

For each specification, your program should print an MCXI-string in a line. Its MCXI-value should be the sum of the two MCXI-values of the MCXI-strings in the specification. No other characters should appear in the output.

Sample Input

```
10
xi x9i
i 9i
c2x2i 4c8x8i
m2ci 4m7c9x8i
9c9x9i i
i 9m9c9x8i
m i
i m
m9i i
9m8c7xi c2x8i
```

Output for the Sample Input

```
3x
x
6cx
5m9c9x9i
m
9m9c9x9i
mi
mi
mx
9m9c9x9i
```

Problem E
Patience

As the proverb says,

> *"Patience is bitter, but its fruit is sweet."*

Writing programs within the limited time may impose some patience on you, but you enjoy it and win the contest, we hope.

The word "patience" has the meaning of perseverance, but it has another meaning in card games. Card games for one player are called "patience" in the UK and "solitaire" in the US.

Let's play a patience in this problem.

In this card game, you use only twenty cards whose face values are positive and less than or equal to 5 (Ace's value is 1 as usual). Just four cards are available for each face value.

At the beginning, the twenty cards are laid in five rows by four columns (See Figure 1). All the cards are dealt face up. An example of the initial layout is shown in Figure 2.

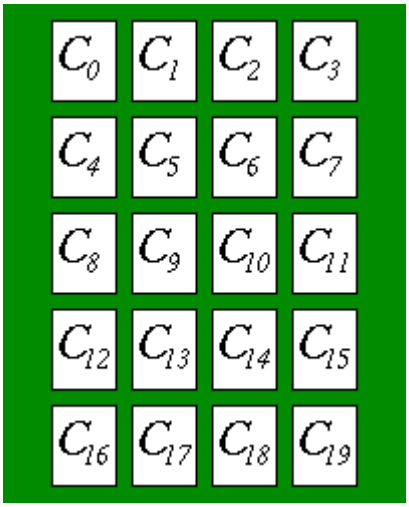


Figure 1: Initial layout

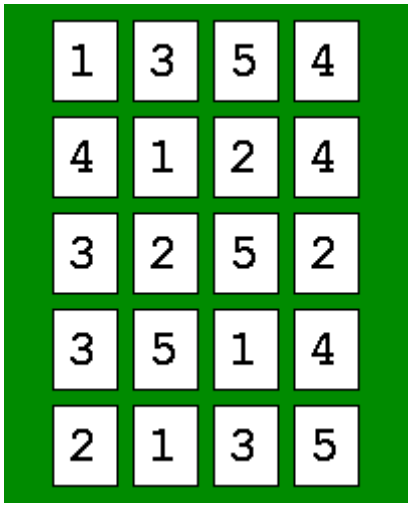


Figure 2: Example of the initial layout

The purpose of the game is to remove as many cards as possible by repeatedly removing a pair of neighboring cards of the same face value. Let us call such a pair a *matching pair*.

The phrase "a pair of neighboring cards" means a pair of cards which are adjacent to each other. For example, in Figure 1, C_6 is adjacent to any of the following eight cards: C_1, C_2, C_3, C_5, C_7, C_9, C_{10} and C_{11}. In contrast, C_3 is adjacent to only the following three cards: C_2, C_6 and C_7.

Every time you remove a pair, you must rearrange the remaining cards as compact as possible.
To put it concretely, each remaining card C_i must be examined in turn in its subscript order to be shifted to the uppermost–leftmost space.

How to play:

1. Search a matching pair.
2. When you find more than one pair, choose one.
   In Figure 3, you decided to remove the pair of C_6 and C_9.

3. Remove the pair. (See Figure 4)
4. Shift the remaining cards to the uppermost-leftmost space (See Figure 5, 6).
5. Repeat the above procedure until you cannot remove any pair.

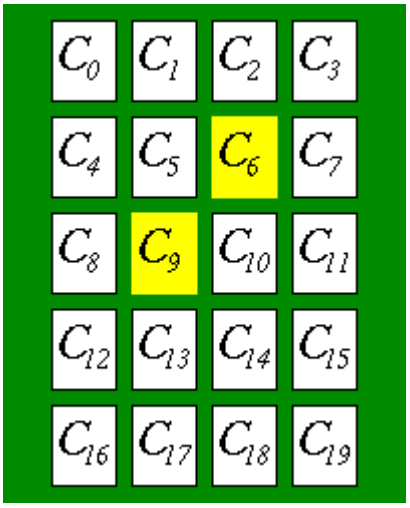


Figure 3: A matching pair found

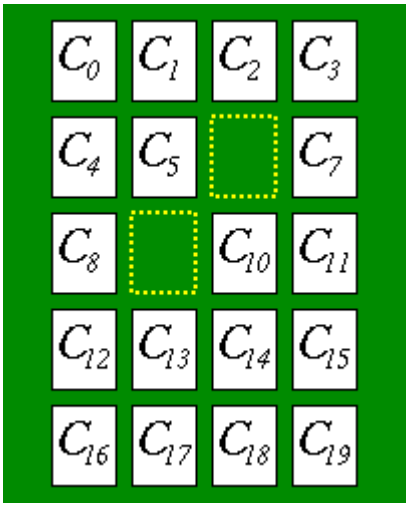


Figure 4: Remove the matching pair

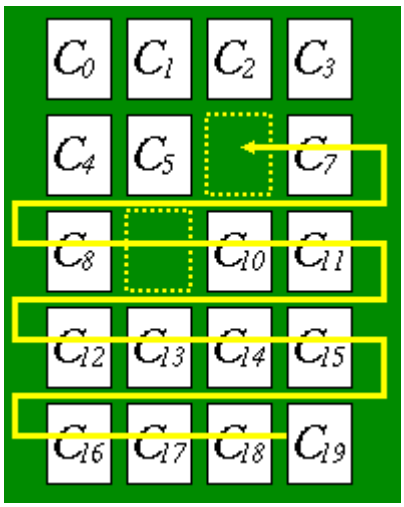


Figure 5: Shift the remaining cards

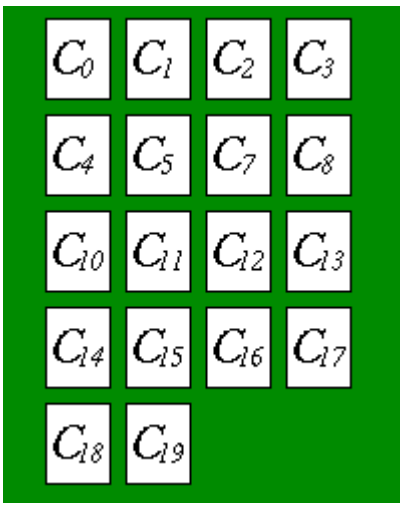


Figure 6: Rearranged layout

If you can remove all the twenty cards, you win the game and your penalty is 0. If you leave some cards, you lose the game and your penalty is the number of the remaining cards.

Whenever you find multiple matching pairs, you must choose one pair out of them as in the step 2 of the above procedure. The result of the game depends on these choices.

Your job is to write a program which answers the minimal penalty for each initial layout.

Input

The input consists of multiple card layouts. The input is given in the following format.

N
$Layout_0$
$Layout_1$
...
$Layout_{N-1}$

N is the number of card layouts. Each card layout gives the initial state of a game. A card layout is given in the following format.

C_0 C_1 C_2 C_3
C_4 C_5 C_6 C_7
C_8 C_9 C_{10} C_{11}
C_{12} C_{13} C_{14} C_{15}
C_{16} C_{17} C_{18} C_{19}

C_i ($0 <= i <= 19$) is an integer from 1 to 5 which represents the face value of the card.

Output

For every initial card layout, the minimal penalty should be output, each in a separate line.

Sample Input

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

Output for the Sample Input

```
0
4
12
0
```

Problem F
Concert Hall Scheduling

You are appointed director of a famous concert hall, to save it from bankruptcy. The hall is very popular, and receives many requests to use its two fine rooms, but unfortunately the previous director was not very efficient, and it has been losing money for many years. The two rooms are of the same size and arrangement. Therefore, each applicant wishing to hold a concert asks for a room without specifying which. Each room can be used for only one concert per day.

In order to make more money, you have decided to abandon the previous fixed price policy, and rather let applicants specify the price they are ready to pay. Each application shall specify a period $[i, j]$ and an asking price w, where i and j are respectively the first and last days of the period ($1 <= i <= j <= 365$), and w is a positive integer in yen, indicating the amount the applicant is willing to pay to use a room for the whole period.

You have received applications for the next year, and you should now choose the applications you will accept. Each application should be either accepted for its whole period or completely rejected. Each concert should use the same room during the whole applied period.

Considering the dire economic situation of the concert hall, artistic quality is to be ignored, and you should just try to maximize the total income for the whole year by accepting the most profitable applications.

Input

The input has multiple data sets, each starting with a line consisting of a single integer n, the number of applications in the data set. Then, it is followed by n lines, each of which represents one application with a period $[i, j]$ and an asking price w yen in the following format.

i j w

A line containing a single zero indicates the end of the input.

The maximum number of applications in a data set is one thousand, and the maximum asking price is one million yen.

Output

For each of the given amount, one line containing a single integer representing the number of combinations of coins should be output. No other characters should appear in the output.

Sample Input

```
4
1 2 10
2 3 10
3 3 10
1 3 10
6
1 20 1000
3 25 10000
5 15 5000
22 300 5500
10 295 9000
7 7 6000
8
32 251 2261
```

```
123 281 1339
211 235 5641
162 217 7273
22 139 7851
194 198 9190
119 274 878
122 173 8640
0
```

Output for the Sample Input

```
30
25500
38595
```