Task 1: MAX

Description. In a sequence of integers, the number of times an integer occurs is called the frequency of the integer. Given a sequence of integers, find the highest frequency.

Input. The input file `MAX.IN` consists of several lines. The first line contains a single integer N ($1 \leq N \leq 10000$) denoting the number of integers in the sequence. Each of the subsequent N lines contains an integer n ($1 \leq n \leq 1000$) in the sequence.

Output. The output file `MAX.OUT` contains an integer that is the highest frequency of the sequence.

Example 1. The input file

```
12
1
2
5
6
3
7
11
345
754
2
5
2
```

gives the 12-integer sequence $1, 2, 5, 6, 3, 7, 11, 345, 754, 2, 5, 2$. Integer 2 occurs 3 times and no other integer occurs more than twice. So 3 is the highest frequency and thus the output file contains the number

```
3
```

Example 2. The input file

```
7
2
4
6
7
7
2
4
```

gives the 7-integer sequence $2, 4, 6, 7, 7, 2, 4$. Integers $2, 4, 7$ each occurs 2 times but integer 6 occurs once. So 2 is the highest frequency and thus the output file contains the number

```
2
```

Task 2: KANSAS

Description. The road map of Kansas, USA, is a grid where there are straight roads North-South and straight roads East-West each 1 mile apart. A truck driver gets instructions in the following form. Drive for two hours north at 45 miles per hour, then for 5 hours west at 65 miles per hour, etc. The driver will have to take a break of 1 hour after 5 hours of continuous driving. The aim is to compute, after how many breaks the driver for the first time passes through or finally returns to the place where she started from.

Example 1. Drive for two hours south at 30 miles per hour, then for four hours west at 40 miles per hour, then for three hours north at 20 miles per hour, then for six hours east at 60 miles per hour, then for five hours north at 30 miles per hour, and then for six hours south at 60 miles per hour.

When following these instructions, the driver will be passing through where she started from after taking two breaks.

Example 2. Drive for one hour south at 30 miles per hour, then for two hours west at 40 miles per hour, then for one hour north at 30 miles per hour, then for one hour east at 79 miles per hour, and then for five hours north at 30 miles per hour.

When following these instructions, the driver will neither pass through nor be back where she started from.

Example 3. Drive for four hours west at 60 miles per hour, then drive for eight hours north at 80 miles per hour, then drive for four hours east at 60 miles per hour, then drive for eight hours south at 100 miles per hour, then drive for ten hours east at 20 miles per hour.

When following these instructions, the driver will pass through where she started from after taking 4 breaks.

Input. The input file `KANSAS.IN` consists of several lines. The first line contains an integer (≤ 1000) which is the number of instructions, followed by one line for each instruction, indicating the direction, the number of hours h ($1 \leq h \leq 200$) and the speed s ($1 \leq s \leq 200$) in miles per hour in this order. The geographical directions are indicated by the position of the small hand on an analog clock; 12 stands for north, 6 for south, 9 for west, and 3 for east.

Thus the input files for the examples will look as follows.

Example 1.

```
6
6 2 30
9 4 40
12 3 20
3 6 60
12 5 30
6 6 60
```

Example 2.

```
5
6 1 30
9 2 40
12 1 30
3 1 79
12 5 30
```

Example 3.

```
5
9 4 60
12 8 80
3 4 60
6 8 100
3 10 20
```

Output. The output file `KANSAS.OUT` contains an integer which is the number of breaks taken before the driver for the first time passes through or finally returns to the place where she started from. If the driver does not reach or pass through the place where she started from after following all instructions, the output file should contain the number -1.

Thus the output files for the examples are as follows.

Example 1.

```
2
```

Example 2.

```
-1
```

Example 3.

```
4
```

Task 3: CARS

Description. On a straight single-lane road, there are n cars parked with a space of d meters wide separating every two adjacent cars. The cars will start to move successively in the same direction as follows.

At the beginning, the first car moves at a speed of v meters per second, while the other cars are standing still. After every 5 seconds from the beginning, every car checks the distance to the car in front or behind, and adapts its speed, if necessary.

The first car adapts as follows. If there is less than 80 meters space to the car behind it, it immediately increases its speed by 5 meters per second; if there is more than 100 meters space to the car behind it, it immediately decreases its speed by 5 meters per second; and otherwise it keeps its speed.

All the other cars adapt as follows: If there is less than 80 meters space to the car in front, it immediately decreases its speed by 5 meters per second; if there is more than 100 meters space to the car in front, it immediately increases its speed by 5 meters per second; and otherwise it keeps its speed.

Against the laws of physics, any change of speed takes immediate effect. If a car stands still and needs to decrease speed, it will remain standing still; the cars never go backwards. A collision occurs, if the space between two cars becomes 0 or less.

The goal is to find out, if a collision has occurred after t 5-second-intervals, and if not, how far the first car will have traveled.

Note that the length of the cars is irrelevant.

Example 1. There are $n = 5$ cars. The first car starts with $v = 10$ meters per second, there are $d = 80$ meters between adjacent cars and there are $t = 3$ 5-second-intervals. **First period:** The first car will travel $10 * 5 = 50$ meters in the first period, while all other cars stand still in this period. **Second period:** At the beginning of the second period, the first car decelerates to a speed of 5 meters per second, because there is now 130 meters space to the car behind, and thus travels $5 * 5 = 25$ meters in the second period. At the beginning of the second period, the second car accelerates to a speed of 5 meters per second, because there is 130 meters space to the car in front, and thus travels $5 * 5 = 25$ meters in the second period. At the beginning of the second period, all other cars remain standing, because there still are 80 meters space to the car in front. **Third period:** At the beginning of the third period, the first car decelerates to a speed of 0 meters per second, because there is again 130 meters space to the car behind, and thus travels $0 * 5 = 0$ meters in the third period. **Answer:** The first car will have traveled $50 + 25 + 0 = 75$ meters after the first three periods.

Example 2. There are $n = 2$ cars. The first car starts with $v = 60$ meters per second, there are $d = 100$ meters between adjacent cars and there are $t = 14$ 5-second-intervals. **Answer:** A collision occurs.

Example 3. There are $n = 20$ cars. The first car starts with $v = 5$ meters per second, there are $d = 5$ meters between adjacent cars and there are $t = 100$ 5-second-intervals. **Answer:** The first car will have traveled 2550 meters.

Input. The input file `CARS.IN` consists of a line containing the four integers n, v, d, and t in this order, and $1 \leq n, v, d, t \leq 1000$.

Thus the input files for the examples will look as follows.

Example 1.

```
5 10 80 3
```

Example 2.

```
2 60 100 14
```

Example 3.

```
20 5 5 100
```

Output. The output file `CARS.OUT` contains the number of meters traveled by the first car if there is no collision. If there is a collision, the output file should contain -1.

Thus the output files for the examples will look as follows.

Example 1.

```
75
```

Example 2.

```
-1
```

Example 3.

```
2550
```

Task 4: ZEROS

Description. The factorial of a positive integer n, written as $n!$, is the product of the first n positive integers. That is,

$$n! = 1 \times 2 \times \cdots \times n$$

Given a positive integer n, find the number of zeros in the decimal representation of $n!$. Of course, leading zeros should not be counted. (Note that decimal representation means base ten representation.)

Example 1. There are 7 zeros in the decimal representation of 20!.

$$20! = 1 \times 2 \times \cdots \times 19 \times 20 = 2432902008176640000$$

Example 2. There are 2 zeros in the decimal representation of 7!.

$$7! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 = 5040$$

Example 3. There is 0 zero in the decimal representation of 4!.

$$4! = 1 \times 2 \times 3 \times 4 = 24$$

Input and Output. The input file `ZEROS.IN` contains a single positive integer $n < 100$.

The output file `ZEROS.OUT` contains a single integer giving the number of zeros in the decimal representation of $n!$.

For Example 1, the input file is

```
20
```

and the output file is

```
7
```

For Example 2, the input file is

```
7
```

and the output file is

```
2
```

For Example 3, the input file is

```
4
```

and the output file is

```
0
```

Task 5: RANK

Description. In a tournament of N players and K games, each game involves 2 players. A player may play any number of games, but in each game his opponent must be a different person. A player needs not play with everybody (it is even possible, though strange, that a player does not play any game at all!). In a game, there is no draw — a player either wins or loses.

After all the games have been played, the players are ranked. There are a few situations where ranking is not possible, but here we are interested only in one particular situation where more than 2 players are involved in a 'cyclic' order. One example is as follows: player A beats player B, player B beats player C, and player C in turns beats player A. In this case, the relative ranking of these 3 players cannot be determined.

Note that a player may be involved in more than one cyclic ordering; when this happens, the player should be counted only once.

(Since we are only interested in players involved in cyclic ordering, those players whose ranking cannot be determined due to other reasons — for instance, a player who did not play any game at all — should not be considered here. See the examples.)

You are given a list of games and their results, and you are to find the *total number of players* whose ranking cannot be determined due to cyclic ordering.

Input. The input file `RANK.IN` consists of several lines.

The first line contains 2 integers N ($2 \leq N \leq 20$) and K ($1 \leq K \leq 30$) in this order, where N is the number of players, and K the number of games played. The players are identified by the integers 1, 2, 3, ..., N.

There are K lines after the first line. Each of the K lines contains 4 integers a b sa sb representing the result of a game: a and b are the identifiers of the players, and sa and sb are the scores of players a and b respectively. All scores are non-negative integers less than 10, and the player with the larger score wins.

The input file below represents the results of a 10-player/12-game tournament:

```
10 12
1 8 2 1
1 2 5 0
10 7 1 2
6 9 6 9
3 4 3 1
9 5 3 1
8 2 6 8
4 9 3 0
4 1 5 2
6 10 3 5
3 5 1 9
6 7 9 8
```

Player 1 beats player 8 in game 1, player 1 beats player 2 in game 2, player 7 beats player 10 in game 3, etc.

Output. The output file `RANK.OUT` contains an integer which is the number of players whose ranking cannot be determined due to cyclic ordering.

In the above example, players 3, 4, 5, 9 are in one cycle, and players 6, 7, 10 in another. The output file should thus contain the number

```
7
```

Example 1. The input file

```
5 3
1 3 9 7
5 1 9 2
3 5 2 0
```

describes a 5-player/3-game tournament. Players 1, 3 and 5 are involved in a cyclic ordering, whereas players 2 and 4 did not play. The output file should contain the number

```
3
```

Example 2. The input file

```
5 6
1 2 2 1
1 5 2 1
1 3 2 1
5 2 0 5
5 3 1 8
2 4 4 2
```

describes a 5-player/6-game tournament. There is no cycle in this case, so the output file contains the number

```
0
```

Example 3. The input file

```
10 5
2 4 0 2
2 6 5 3
8 2 8 2
6 4 6 2
8 6 0 2
```

describes a 10-player/5-game tournament. There are 2 cycles: players 2, 4, 6 are involved in one cycle, while players 2, 6, 8 are in another. So players $2, 4, 6, 8$ are involved in cyclic ordering and the output file should contain the number

```
4
```

Task 6: FAX

Description. You are asked to design a compression scheme for transmitting fax images of documents. After an image is scanned in using a scanner, you obtain a sequence of numbers where each number x takes a value from 1 to 256. For example, an image may be converted into a sequence of the form

$$1, 1, 1, 1, 1, 46, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 25, 26, 25, 25, 255, 256, 256, 256, 255, \cdots$$

Since it is not necessary to transmit the exact image, you decide to discretize each number to 4 levels, namely 1, 86, 172 and 256, so that you can code each level using only 2 bits as follows

Level	1	86	172	256
Code	00	01	10	11

Then you realize that large areas of the same value (e.g. white) often occur in documents resulting in long sequences of very similar numbers. You decide to adopt a compression scheme as follows:

1. For the first number in the sequence, you will use the 2 bits as described previously.
2. For all other numbers:
   - if the previous number is the same as the current number, you will transmit the bit 0.
   - Otherwise, you will transmit the bit 1 followed by the 2-bit code of the number. That is,
     - 1 followed by 00 for number 1,
     - 1 followed by 01 for the number 86,
     - 1 followed by 10 for the number 172,
     - 1 followed by 11 for the number 256.

Finally, since you are not worried about a small amount of error, you realize that you can improve your scheme by ignoring isolated changes. For example, if the input sequence is $2, 2, 2, 2, 2, 46, 2, 2$ and you transmit the sequence $1, 1, 1, 1, 1, 86, 1, 1$, the total error will be $1 + 1 + 1 + 1 + 1 + 40 + 1 + 1 = 47$ and the string that needs to be transmitted is 0000001011000. If, instead you choose to transmit the sequence $1, 1, 1, 1, 1, 1, 1, 1$, the total error will be $1 + 1 + 1 + 1 + 1 + 45 + 1 + 1 = 52$ but the string that needs to be transmitted would be 000000000 which is shorter.

You decide to measure the overall cost of a transmission by the following weighted sum
cost = total error + W(total number of bits transmitted)*,
where

1. the weight W can be adjusted to give more emphasis to either the total error or the total number of bits transmitted
2. for an input sequence $x_1, x_2, \ldots, x_N$ and a transmitted sequence $y_1, y_2, \ldots, y_N$, both of length N, the total error is $\sum_{i=1}^{N} |x_i - y_i| = |x_1 - y_1| + \cdots + |x_N - y_N|$.

Example. For the input sequence $2, 2, 2, 2, 2, 46, 2, 2$ with $W = 100$,

- we can transmit the sequence $y_1, \ldots, y_8 = 1, 1, 1, 1, 1, 86, 1, 1$ by transmitting the string 0000001011000 with a cost of $47 + 100 * 13 = 1347$;
- or we can transmit the sequence $y_1, \ldots, y_8 = 1, 1, 1, 1, 1, 1, 1, 1$ by transmitting the string 000000000 with a cost of $52 + 100 * 9 = 952$.

In this case the second option has lower cost.

Input. The input file `FAX.IN` consists of several lines. The first line contains two integers N ($1 \leq N \leq 50$) and W ($1 \leq W \leq 100$) in this order. Integer N denotes the length of the input sequence and integer W is the weight. Each of the subsequent N lines contains an integer x ($1 \leq x \leq 256$) of the sequence. For example, the input file

```
8 100
2
2
2
2
2
46
2
2
```

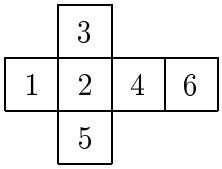
represents the 8-number sequence $2, 2, 2, 2, 2, 46, 2, 2$ with $W = 100$.

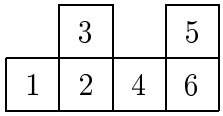
Output. The output file `FAX.OUT` contains a single integer which is the smallest possible cost required to transmit the input sequence with the given value of W. For the input sequence given in the above example, the output file should contain the number

```
952
```

Task 7: CUBE

Description. Folding six squares connected in some special ways can form a cube. For example, in the diagram below, the six squares on the left can be folded into a cube (with face 1 opposite face 4, face 2 opposite face 6, and face 3 opposite face 5) but the six squares on the right cannot be folded into a cube (faces 3 and 5 overlap).





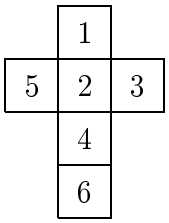
A 6×6 array is used to represent the six squares. Only 6 of the 36 cells are used to represent the 6 squares. A cell representing a square contains the number for the square. Other cells contain the number 0. Note that the same 6 squares can be represented in many ways. For example, the invalid six squares of the above example can be represented as follows (the one on the right is obtained after a 90 degree rotation of the one on the left):

```
0 0 0 0 0 0
0 3 0 5 0 0
1 2 4 6 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 5 6
0 0 0 0 0 4
0 0 0 0 3 2
0 0 0 0 0 1
```

Given a square representation, determine if the squares can be folded into a cube; if so, find the face opposite face 1.

Input. The input file `CUBE.IN` consists of six lines with each line containing six integers. All but six of the input integers are zeros. The non-zero integers are $1, 2, 3, 4, 5, 6$.

For example, the input file for the squares 

may look like

```
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
0 5 2 3 0 0
0 0 4 0 0 0
0 0 6 0 0 0
```

Output. The output file `CUBE.OUT` consists of a single integer. The integer is 0 if the squares cannot be folded into a cube; otherwise, the integer is the number of the face opposite face 1.

For the above example, the output file should contain the number

4