357 Let Me Count The Ways

After making a purchase at a large department store, Mel's change was 17 cents. He received 1 dime, 1 nickel, and 2 pennies. Later that day, he was shopping at a convenience store. Again his change was 17 cents. This time he received 2 nickels and 7 pennies. He began to wonder ' "How many stores can I shop in and receive 17 cents change in a different configuration of coins? After a suitable mental struggle, he decided the answer was 6. He then challenged you to consider the general problem.

Write a program which will determine the number of different combinations of US coins (penny, nickel, dime, quarter, half-dollar) which may be used to produce a given amount of money.

Input

The input will consist of a set of numbers between 0 and 99 inclusive, one per line in the input file.

Output

The output will consist of the appropriate statement from the selection below on a single line in the output file for each input value. The number m is the number your program computes, n is the input value.

```
There are  m ways to produce  n cents change.
There is only 1 way to produce  n cents change.
```

Sample input

```
17
11
4
```

Sample output

```
There are 6 ways to produce 17 cents change.
There are 4 ways to produce 11 cents change.
There is only 1 way to produce 4 cents change.
```

Problem F

TV game

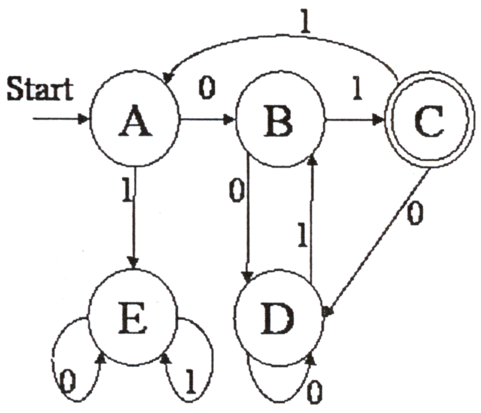


Figure 1.1 - The carpet for one game.

The next TV game will be played by single players on a special kind of labyrinth. The player will step on a carpet with a drawing like the one in fig. 1.1, and wait on position A. Each position has two ways out, labeled by **0** and **1**, which lead to the next position. To choose which way to take, the player must answer a question. If the answer is correct he takes the **1** way, otherwise the **0** way is followed. Of course, the answer may be deliberately wrong if the **0** way is sought for. The next position may be different or remain the same as before.

Some of the positions, indicated by a double circle, are special. If, exactly after a predetermined number of moves, the player gets on one of those special positions he wins, otherwise he loses.

In the example, if the total number of moves is **m**=2, failing the first question and passing the second, i. e. the sequence **01**, directs the player to go from **A**, the start position, to B and then to C. It solves the problem, as **C** is a special position, in the sole possible way. In fact, **00** would lead to **D** and **10** and **11** to **E**, which are not special. In the case **m**=3, there is no solution. But in the case **m**=5, several solutions are available, for instance **01011**, **01101** or **00011**. Thus there are 3 out of 2^**m** = 32 ways to win, which gives an idea of the probability of winning just choosing the moves by tossing a coin.

Notice that should **A** also be a special position, there would be a way of scoring in zero moves.

Problem

The problem to be solved is, given a carpet and a number of moves **m**, to determine the number of different ways to score, i.e., to reach one of the special positions in exactly **m** moves, from the start position. The start position is the first position, labeled **A**. From each position there are exactly 2 ways out, labeled by the symbols **0** and **1**.

Input

The input is a text file with one or more test cases, each of them containing several lines as follows.

The first line of the input contains the number **N** (integer format) of positions. The positions are labeled in alphabetic sequence, starting from **A**, and there are at most 26. The next **N** lines contain four characters each, separated by single spaces, where the first is the name of a position, the second the position the player reaches if he chooses the path labeled **0**, the third the position the player reaches if he chooses the path labeled **1**, and the fourth a 'x' if the position is special or a '-' if not.

The last line specifies **m**, the number of moves to be considered, **$0 \leq m \leq 30$**.

Output

For each test case, the output consists of one line which contains one integer indicating the number of different ways to win. **0** means there are no solutions.

Sample Input

```
5
A B E -
B D C -
C D A x
D D B -
E E E -
5
```

Sample Output

```
3
```

Gabriel David, MIUP'2003
(Portuguese National ACM Programming Contest)

231 Testing the CATCHER

A military contractor for the Department of Defense has just completed a series of preliminary tests for a new defensive missile called the CATCHER which is capable of intercepting multiple incoming offensive missiles. The CATCHER is supposed to be a remarkable defensive missile. It can move forward, laterally, and downward at very fast speeds, and it can intercept an offensive missile without being damaged. But it does have one major flaw. Although it can be fired to reach any initial elevation, it has no power to move higher than the last missile that it has intercepted.

The tests which the contractor completed were computer simulations of battlefield and hostile attack conditions. Since they were only preliminary, the simulations tested only the CATCHER's vertical movement capability. In each simulation, the CATCHER was fired at a sequence of offensive missiles which were incoming at fixed time intervals. The only information available to the CATCHER for each incoming missile was its height at the point it could be intercepted and where it appeared in the sequence of missiles. Each incoming missile for a test run is represented in the sequence only once.

The result of each test is reported as the sequence of incoming missiles and the total number of those missiles that are intercepted by the CATCHER in that test.

The General Accounting Office wants to be sure that the simulation test results submitted by the military contractor are attainable, given the constraints of the CATCHER. You must write a program that takes input data representing the pattern of incoming missiles for several different tests and outputs the maximum numbers of missiles that the CATCHER can intercept for those tests. For any incoming missile in a test, the CATCHER is able to intercept it if and only if it satisfies one of these two conditions:

1. The incoming missile is the first missile to be intercepted in this test.

-or-

2. The missile was fired after the last missile that was intercepted and it is not higher than the last missile which was intercepted.

Input

The input data for any test consists of a sequence of one or more non-negative integers, all of which are less than or equal to 32,767, representing the heights of the incoming missiles (the test pattern). The last number in each sequence is `-1`, which signifies the end of data for that particular test and is not considered to represent a missile height. The end of data for the entire input is the number `-1` as the first value in a test; it is not considered to be a separate test.

Output

Output for each test consists of a test number (`Test #1`, `Test #2`, etc.) and the maximum number of incoming missiles that the CATCHER could possibly intercept for the test. That maximum number appears after an identifying message. There must be at least one blank line between output for successive data sets.

Note: The number of missiles for any given test is not limited. If your solution is based on an inefficient algorithm, it **may not** execute in the allotted time.

Sample Input

```
389
207
155
300
299
170
158
65
-1
23
34
21
-1
-1
```

Sample Output

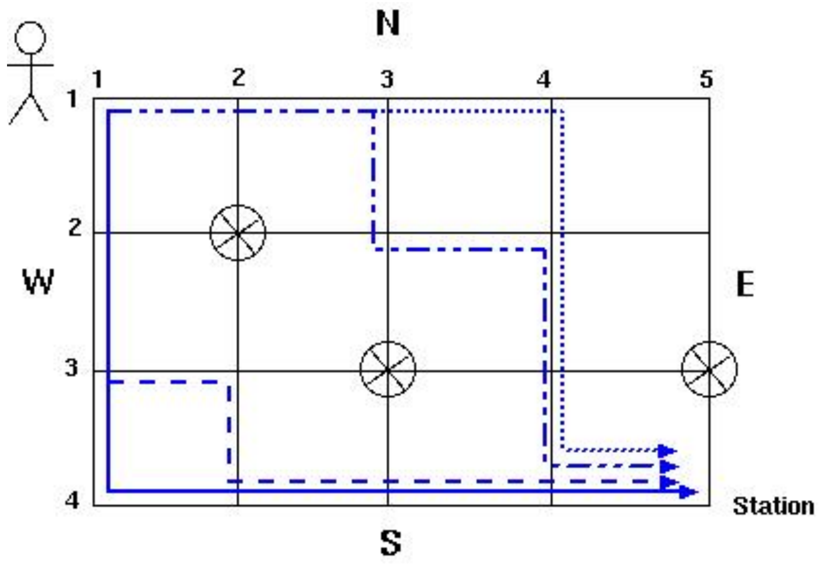
```
Test #1:
  maximum possible interceptions: 6

Test #2:
  maximum possible interceptions: 2
```

Walking on the Safe Side

Square City is a very easy place for people to walk around. The two-way streets run North-South or East-West dividing the city into regular blocks. Most street intersections are safe for pedestrians to cross. In some of them, however, crossing is not safe and pedestrians are forced to use the available underground passages. Such intersections are avoided by walkers. The entry to the city park is on the North-West corner of town, whereas the railway station is on the South-East corner.

Suppose you want to go from the park to the railway station, and do not want to walk more than the required number of blocks. You also want to make your way avoiding the underground passages, that would introduce extra delay. Your task is to determine the number of different paths that you can follow from the park to the station, satisfying both requirements.

The example in the picture illustrates a city with 4 E-W streets and 5 N-S streets. Three intersections are marked as unsafe. The path from the park to the station is 3 + 4 = 7 blocks long and there are 4 such paths that avoid the underground passages.



Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described

below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of the input contains the number of East-West streets ***W*** and the number of North-South streets ***N***. Each one of the following ***W*** lines starts with the number of an East-West street, followed by zero or more numbers of the North-South crossings which are unsafe. Streets are numbered from 1.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The number of different minimal paths from the park to the station avoiding underground passages.

Sample Input

```
1

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

Sample Output

```
4
```

Cristina Ribeiro, MIUP'2001

Problem B

Flight Planner

Input: standard input
Output: standard output
Time Limit: 1 second
Memory Limit: 32 MB

Calculating the minimal cost for a flight involves calculating an optimal flight-altitude depending on wind-strengths changing with different altitudes. It's not enough just to ask for the route with optimal wind-strength, because due to the mass of a plane you need a certain amount of fuel to rise. Moreover due to safety regulations it's forbidden to fly above a certain altitude and you can't fly under zero-level.

In order to simplify the problem for now, we assume that for each 100 miles of flight you have only three possiblities: to climb one mile, to hold your altitude or to sink one mile.

Climb flight requires 60 units of fuel, holding your altitude requires 30 units and sinking requires 20 units.

In the case of headwind you need more fuel while you can save fuel flying with tailwind. Windstrength w will satisfy the condition -10 <= w <= 10, where negative windstrength is meant to be headwind and positive windstrength is tailwind.

For one unit of tailwind you can save one unit of fuel each 100 miles; each unit of headwind will cost an extra unit of fuel.

For example to climb under conditions of windstrength w = -5, you need 65 units of fuel for this 100 miles.
Given the windstrengths on different altitudes for a way from here to X, calculate the minimal amount of fuel you need to fly to X.

Input

The first line of the input file contains the number N of test cases in the file. The first line of each test case contains a single integer X, the distance to fly, with 1 <= X <= 100000 miles and X is a multiple of 100. Notice that it's not allowed to fly higher than 9 miles over zero and that you have to decide whether to climb, hold your altitude or to sink only for every 100 miles.
For every mile of allowed altitude (starting at altitude 9 down to altitude 0) there follow X/100 windstrengths, starting with the windstrength at your current position up to the windstrength at position X-100 in steps of 100 miles. Test cases are separated by one or more blank lines.

Output

For each test case output the minimal amount of fuel used flying from your current position (at altitude 0) to X (also at altitude 0), followed by a blank line.

Sample Input

```
3

400
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
1 9 9 1
1 -9 -9 1

1000
 9  9  9  9  9  9  9  9  9  9
 9  9  9  9  9  9  9  9  9  9
 9  9  9  9  9  9  9  9  9  9
 9  9  9  9  9  9  9  9  9  9
 9  9  9  9  9  9  9  9  9  9
 9  9  9  9  9  9  9  9  9  9
 7  7  7  7  7  7  7  7  7  7
-5 -5 -5 -5 -5 -5 -5 -5 -5 -5
-7 -3 -7 -7 -7 -7 -7 -7 -7 -7
-9 -9 -9 -9 -9 -9 -9 -9 -9 -9
```

Sample Output

```
120

354
```

Frank Hutter

Problem D	Bar Codes
Time Limit	1 Second

A bar-code symbol consists of alternating dark and light bars, starting with a dark bar on the left. Each bar is a number of units wide. Figure 1 shows a bar-code symbol consisting of 4 bars that extend over 1+2+3+1=7 units.

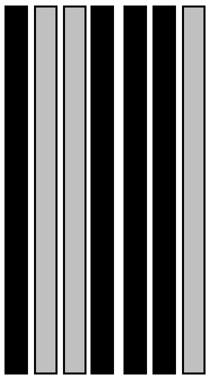


Figure 1: Bar-code over 7 units with 4 bars

In general, the bar code **BC(n,k,m)** is the set of all symbols with **k** bars that together extend over exactly **n** units, each bar being at most **m** units wide. For instance, the symbol in Figure 1 belongs to BC(7,4,3) but not to BC(7,4,2). Figure 2 shows all 16 symbols in BC(7,4,3). Each '1' represents a dark unit, each '0' a light unit.

```
0: 1000100 | 4: 1001110 | 8:  1100100 | 12: 1101110
1: 1000110 | 5: 1011000 | 9:  1100110 | 13: 1110010
2: 1001000 | 6: 1011100 | 10: 1101000 | 14: 1110100
3: 1001100 | 7: 1100010 | 11: 1101100 | 15: 1110110
```

Figure 2: All symbols of BC(7,4,3)

Input

Each input will contain three positive integers **n**, **k**, and **m** ($1 \le$ **n**, **k**, **m** ≤ 50).

Output

For each input print the total number of symbols in **BC(n,k,m)**. Output will fit in 64-bit signed integer.

Sample Input	Output for Sample Input
7 4 3 7 4 2	16 4

Collected (Slightly Modified by Md. Kamruzzaman)