Problem A: Keeps Going and Going and ...

Lazy functional languages like Haskell and Miranda support features that are not found in other programming languages, including infinite lists. Consider the following simple (and useful) recursive declaration:

```
letrec
  count n = cons n (count (n+1))
in
  count 0
```

The function `cons` *cons*tructs lists, so the above declaration creates the following structure:

```
  cons 0 (count 1)
= cons 0 (cons 1 (count 2))
= cons 0 (cons 1 (cons 2 ...))
= [0,1,2,...]
```

Lazy languages can do this because they only evaluate expressions that are actually used. If a program creates an infinite list and only looks at items 2 and 3 in it, the values in positions 0 and 1 are never evaluated and the list structure is only evaluated so far as the fourth node.

It is also possible to use more than one function to build an infinite list. Here is a declaration that creates the list `["even","odd","even",...]`:

```
letrec
  even = cons "even" odd
  odd = cons "odd" even
in
  even
```

There are also functions that manipulate infinite lists. The functions `take` and `drop` can be used to remove elements from the start of the list, returning the (removed) front elements or the remainder of the list, respectively. Another useful function is `zip`, which combines two lists like the slider on a zipper combines the teeth. For example,

```
zip (count 0) (count 10) = [0,10,1,11,2,12,...]
```

Your task is to implement a subset of this functionality.

Input

The first line of input consists of two positive integers, n and m. n is the number of declarations to follow and m is the number of test cases.

Each declaration takes the form *name* `=` *expr*. There are two forms for *expr*: `zip` *name1 name2* and $x_0\ x_1\ \ldots\ x_i$ *name3*. In the first case, *name* is the result of zipping *name1* and *name2* together. The other case defines the first $i+1$ non-negative integers in the list *name* (where i is at least 0) and *name3* is the name of the list that continues it (mandatory—all lists will be infinite).

The test cases take the form *name* s e, where s and e are non-negative integers, $s \leq e$ and $e - s < 250$.

No line of input will be longer than 80 characters. Names consist of a single capital letter.

Output

For each test case, print the integers in positions s to e of the list *name*. List elements are numbered starting with 0.

Sample Input

```
5 3
S = 4 3 2 1 A
O = 1 O
E = 0 E
A = zip E O
Z = zip Z S
A 43455436 43455438
S 2 5
Z 1 10
```

Sample Output

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

Problem B: Scheduling Lectures

You are teaching a course and must cover n ($1 \leq n \leq 1000$) topics. The length of each lecture is L ($1 \leq L \leq 500$) minutes. The topics require $t_1, t_2, \ldots, t_n$ ($1 \leq t_i \leq L$) minutes each. For each topic, you must decide in which lecture it should be covered. There are two scheduling restrictions:

1. Each topic must be covered in a single lecture. It cannot be divided into two lectures. This reduces discontinuity between lectures.
2. Topic i must be covered before topic $i+1$ for all $1 \leq i < n$. Otherwise, students may not have the prerequisites to understand topic $i+1$.

With the above restrictions, it is sometimes necessary to have free time at the end of a lecture. If the amount of free time is at most 10 minutes, the students will be happy to leave early. However, if the amount of free time is more, they would feel that their tuition fees are wasted. Therefore, we will model the dissatisfaction index (DI) of a lecture by the formula:

$$DI = \begin{cases} 0 & \text{if } t = 0, \\ -C & \text{if } 1 \leq t \leq 10, \\ (t-10)^2 & \text{otherwise,} \end{cases}$$

where C is a positive integer, and t is the amount of free time at the end of a lecture. The total dissatisfaction index is the sum of the DI for each lecture.

For this problem, you must find the minimum number of lectures that is needed to satisfy the above constraints. If there are multiple lecture schedules with the minimum number of lectures, also minimize the total dissatisfaction index.

Input

The input consists of a number of cases. The first line of each case contains the integer n, or 0 if there are no more cases. The next line contains the integers L and C. These are followed by n integers $t_1, t_2, \ldots, t_n$.

Output

For each case, print the case number, the minimum number of lectures used, and the total dissatisfaction index for the corresponding lecture schedule on three separate lines. Output a blank line between cases.

Sample Input

```
6
30 15
10
10
10
10
10
10
10
120 10
80
80
10
50
30
20
40
30
120
100
0
```

Sample Output

```
Case 1:

Minimum number of lectures: 2
Total dissatisfaction index: 0

Case 2:

Minimum number of lectures: 6
Total dissatisfaction index: 2700
```

Problem C: Counterfeit Dollar

Sally Jones has a dozen Voyageur silver dollars. However, only eleven of the coins are true silver dollars; one coin is counterfeit even though its color and size make it indistinguishable from the real silver dollars. The counterfeit coin has a different weight from the other coins but Sally does not know if it is heavier or lighter than the real coins.

Happily, Sally has a friend who loans her a very accurate balance scale. The friend will permit Sally three weighings to find the counterfeit coin. For instance, if Sally weighs two coins against each other and the scales balance then she knows these two coins are true. Now if Sally weighs one of the true coins against a third coin and the scales do not balance then Sally knows the third coin is counterfeit and she can tell whether it is light or heavy depending on whether the balance on which it is placed goes up or down, respectively.

By choosing her weighings carefully, Sally is able to ensure that she will find the counterfeit coin with exactly three weighings.

Input

The first line of input is an integer n ($n > 0$) specifying the number of cases to follow. Each case consists of three lines of input, one for each weighing. Sally has identified each of the coins with the letters A–L. Information on a weighing will be given by two strings of letters and then one of the words "up", "down", or "even". The first string of letters will represent the coins on the left balance; the second string, the coins on the right balance. (Sally will always place the same number of coins on the right balance as on the left balance.) The word in the third position will tell whether the right side of the balance goes up, down, or remains even.

Output

For each case, the output will identify the counterfeit coin by its letter and tell whether it is heavy or light. The solution will always be uniquely determined.

Sample Input

```
1
ABCD EFGH even
ABCI EFJK up
ABIJ EFGH even
```

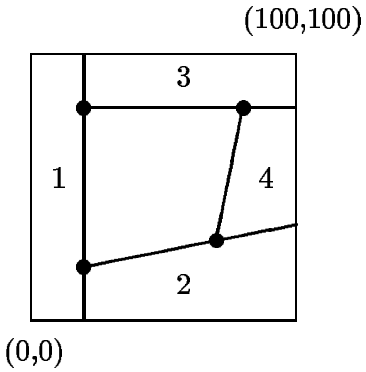
Sample Output

```
K is the counterfeit coin and it is light.
```

Problem D: Metal Cutting

In order to build a ship to travel to Eindhoven, The Netherlands, various sheet metal parts have to be cut from rectangular pieces of sheet metal. Each part is a convex polygon with at most 8 vertices. Each rectangular piece of sheet metal has width n and height m, so that the four corners of the sheet can be specified by the Cartesian coordinates $(0,0)$, $(0,m)$, (n,m), and $(n,0)$ in clockwise order. The cutting machine available can make only straight-line cuts completely through the metal. That is, it cannot cut halfway through the sheet, turn, and then cut some more. You are asked to write a program to determine the minimum total length of cuts this machine has to make in order to cut out the polygon.

For example, if $n = m = 100$, and the polygon has vertices $(80,80)$, $(70,30)$, $(20,20)$, and $(20,80)$, the following diagram shows the optimal cut (the thick lines). The numbers show the order in which the cuts are made.



Input

The first line of input contains the two integers n and m where $0 < n, m \leq 500$. The next line contains p, the number of vertices in the polygon, where $3 \leq p \leq 8$. Each of the next p lines contains two integers x and y where $0 < x < n$ and $0 < y < m$, specifying the vertices of the polygon. The vertices are listed in clockwise order. You may assume that the polygon does not intersect itself, and that no three consecutive vertices are colinear.

Output

Print the minimum total length of cuts required to cut out the given polygon, accurate to 3 decimal places.

Sample Input

```
100 100
4
80 80
```

```
70 30
20 20
20 80
```

Sample Output

```
Minimum total length = 312.575
```