Codeforces Beta Round #74 (Div. 2 Only)

A. Cableway

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

A group of university students wants to get to the top of a mountain to have a picnic there. For that they decided to use a cableway.

A cableway is represented by some cablecars, hanged onto some cable stations by a cable. A cable is scrolled cyclically between the first and the last cable stations (the first of them is located at the bottom of the mountain and the last one is located at the top). As the cable moves, the cablecar attached to it move as well.

The number of cablecars is divisible by three and they are painted three colors: red, green and blue, in such manner that after each red cablecar goes a green one, after each green cablecar goes a blue one and after each blue cablecar goes a red one. Each cablecar can transport no more than two people, the cablecars arrive with the periodicity of one minute (i. e. every minute) and it takes exactly 30 minutes for a cablecar to get to the top.

All students are divided into three groups: r of them like to ascend only in the red cablecars, g of them prefer only the green ones and b of them prefer only the blue ones. A student never gets on a cablecar painted a color that he doesn't like,

The first cablecar to arrive (at the moment of time 0) is painted red. Determine the least time it will take all students to ascend to the mountain top.

Input

The first line contains three integers r, g and b ($0 \le r, g, b \le 100$). It is guaranteed that $r + g + b > 0$, it means that the group consists of at least one student.

Output

Print a single number — the minimal time the students need for the whole group to ascend to the top of the mountain.

Sample test(s)

input
1 3 2
output
34

input
3 2 1
output
33

Note

Let's analyze the first sample.

At the moment of time 0 a red cablecar comes and one student from the r group get on it and ascends to the top at the moment of time 30.

At the moment of time 1 a green cablecar arrives and two students from the g group get on it; they get to the top at the moment of time 31.

At the moment of time 2 comes the blue cablecar and two students from the b group get on it. They ascend to the top at the moment of time 32.

At the moment of time 3 a red cablecar arrives but the only student who is left doesn't like red and the cablecar leaves empty.

At the moment of time 4 a green cablecar arrives and one student from the g group gets on it. He ascends to top at the moment of time 34.

Thus, all the students are on the top, overall the ascension took exactly 34 minutes.

B. African Crossword

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

An African crossword is a rectangular table $n \times m$ in size. Each cell of the table contains exactly one letter. This table (it is also referred to as grid) contains some encrypted word that needs to be decoded.

To solve the crossword you should cross out all repeated letters in rows and columns. In other words, a letter should only be crossed out if and only if the corresponding column or row contains at least one more letter that is exactly the same. Besides, all such letters are crossed out simultaneously.

When all repeated letters have been crossed out, we should write the remaining letters in a string. The letters that occupy a higher position follow before the letters that occupy a lower position. If the letters are located in one row, then the letter to the left goes first. The resulting word is the answer to the problem.

You are suggested to solve an African crossword and print the word encrypted there.

Input

The first line contains two integers n and m ($1 \le n, m \le 100$). Next n lines contain m lowercase Latin letters each. That is the crossword grid.

Output

Print the encrypted word on a single line. It is guaranteed that the answer consists of at least one letter.

Sample test(s)

input
3 3 cba bcd cbc
output
abcd

input
5 5 fcofd ooedo afaoa rdcdf eofsf
output
codeforces

C. Robbery

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

It is nighttime and Joe the Elusive got into the country's main bank's safe. The safe has n cells positioned in a row, each of them contains some amount of diamonds. Let's make the problem more comfortable to work with and mark the cells with positive numbers from 1 to n from the left to the right.

Unfortunately, Joe didn't switch the last security system off. On the plus side, he knows the way it works.

Every minute the security system calculates the total amount of diamonds for each two adjacent cells (for the cells between whose numbers difference equals 1). As a result of this check we get an n - 1 sums. If at least one of the sums differs from the corresponding sum received during the previous check, then the security system is triggered.

Joe can move the diamonds from one cell to another between the security system's checks. He manages to move them no more than m times between two checks. One of the three following operations is regarded as moving a diamond: moving a diamond from any cell to any other one, moving a diamond from any cell to Joe's pocket, moving a diamond from Joe's pocket to any cell. Initially Joe's pocket is empty, and it can carry an unlimited amount of diamonds. It is considered that before all Joe's actions the system performs at least one check.

In the morning the bank employees will come, which is why Joe has to leave the bank before that moment. Joe has only k minutes left before morning, and on each of these k minutes he can perform no more than m operations. All that remains in Joe's pocket, is considered his loot.

Calculate the largest amount of diamonds Joe can carry with him. Don't forget that the security system shouldn't be triggered (even after Joe leaves the bank) and Joe should leave before morning.

Input

The first line contains integers n, m and k ($1 \le n \le 10^4$, $1 \le m, k \le 10^9$). The next line contains n numbers. The i-th number is equal to the amount of diamonds in the i-th cell — it is an integer from 0 to 10^5.

Output

Print a single number — the maximum number of diamonds Joe can steal.

Sample test(s)

input
2 3 1 2 3

output
0

input
3 2 2 4 1 3

output
2

Note

In the second sample Joe can act like this:

The diamonds' initial positions are $4\ 1\ 3$.

During the first period of time Joe moves a diamond from the 1-th cell to the 2-th one and a diamond from the 3-th cell to his pocket.

By the end of the first period the diamonds' positions are $3\ 2\ 2$. The check finds no difference and the security system doesn't go off.

During the second period Joe moves a diamond from the 3-rd cell to the 2-nd one and puts a diamond from the 1-st cell to his pocket.

By the end of the second period the diamonds' positions are $2\ 3\ 1$. The check finds no difference again and the security system doesn't go off.

Now Joe leaves with 2 diamonds in his pocket.

D. Widget Library

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

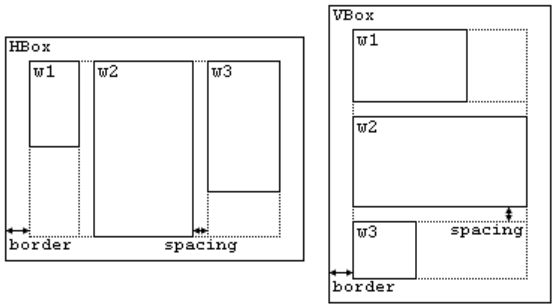
Vasya writes his own library for building graphical user interface. Vasya called his creation `VTK` (`VasyaToolKit`). One of the interesting aspects of this library is that widgets are packed in each other.

A widget is some element of graphical interface. Each widget has width and height, and occupies some rectangle on the screen. Any widget in Vasya's library is of type `Widget`. For simplicity we will identify the widget and its type.

Types `HBox` and `VBox` are derivatives of type `Widget`, so they also are types `Widget`. Widgets `HBox` and `VBox` are special. They can store other widgets. Both those widgets can use the `pack()` method to pack directly in itself some other widget. Widgets of types `HBox` and `VBox` can store several other widgets, even several equal widgets — they will simply appear several times. As a result of using the method `pack()` only the link to the packed widget is saved, that is when the packed widget is changed, its image in the widget, into which it is packed, will also change.

We shall assume that the widget a is packed in the widget b if there exists a chain of widgets $a = c_1, c_2, ..., c_k = b$, $k \geq 2$, for which c_i is packed directly to c_{i+1} for any $1 \leq i < k$. In Vasya's library the situation when the widget a is packed in the widget a (that is, in itself) is not allowed. If you try to pack the widgets into each other in this manner immediately results in an error.

Also, the widgets `HBox` and `VBox` have parameters `border` and `spacing`, which are determined by the methods `set_border()` and `set_spacing()` respectively. By default both of these options equal 0.



The picture above shows how the widgets are packed into `HBox` and `VBox`. At that `HBox` and `VBox` automatically change their size depending on the size of packed widgets. As for `HBox` and `VBox`, they only differ in that in `HBox` the widgets are packed horizontally and in `VBox` — vertically. The parameter `spacing` sets the distance between adjacent widgets, and `border` — a frame around all packed widgets of the desired width. Packed widgets are placed exactly in the order in which the `pack()` method was called for them. If within `HBox` or `VBox` there are no packed widgets, their sizes are equal to 0×0, regardless of the options `border` and `spacing`.

The construction of all the widgets is performed using a scripting language `VasyaScript`. The description of the language can be found in the input data.

For the final verification of the code Vasya asks you to write a program that calculates the sizes of all the widgets on the source code in the language of `VasyaScript`.

Input

The first line contains an integer n — the number of instructions ($1 \leq n \leq 100$). Next n lines contain instructions in the language `VasyaScript` — one instruction per line. There is a list of possible instructions below.

- "`Widget [name]([x],[y])`" — create a new widget `[name]` of the type `Widget` possessing the width of `[x]` units and the height of `[y]` units.
- "`HBox [name]`" — create a new widget `[name]` of the type `HBox`.
- "`VBox [name]`" — create a new widget `[name]` of the type `VBox`.
- "`[name1].pack([name2])`" — pack the widget `[name2]` in the widget `[name1]`. At that, the widget `[name1]` must be of type `HBox` or `VBox`.
- "`[name].set_border([x])`" — set for a widget `[name]` the `border` parameter to `[x]` units. The widget `[name]` must be of type `HBox` or `VBox`.
- "`[name].set_spacing([x])`" — set for a widget `[name]` the `spacing` parameter to `[x]` units. The widget `[name]` must be of type `HBox` or `VBox`.

All instructions are written without spaces at the beginning and at the end of the string. The words inside the instruction are separated by exactly one space. There are no spaces directly before the numbers and directly after them.

The case matters, for example, "`wiDget x`" is not a correct instruction. The case of the letters is correct in the input data.

All names of the widgets consist of lowercase Latin letters and has the length from 1 to 10 characters inclusive. The names of all widgets are pairwise different. All numbers in the script are integers from 0 to 100 inclusive

It is guaranteed that the above-given script is correct, that is that all the operations with the widgets take place after the widgets are created and no widget is packed in itself. It is guaranteed that the script creates at least one widget.

Output

For each widget print on a single line its name, width and height, separated by spaces. The lines must be ordered lexicographically by a widget's name.

Please, do not use the `%lld` specificator to read or write 64-bit integers in C++. It is preferred to use `cout` stream (also you may use `%I64d` specificator)

Sample test(s)

input

```
12
Widget me(50,40)
VBox grandpa
HBox father
grandpa.pack(father)
father.pack(me)
grandpa.set_border(10)
grandpa.set_spacing(20)
Widget brother(30,60)
father.pack(brother)
Widget friend(20,60)
Widget uncle(100,20)
grandpa.pack(uncle)
```

output

```
brother 30 60
father 80 60
friend 20 60
grandpa 120 120
me 50 40
uncle 100 20
```

input

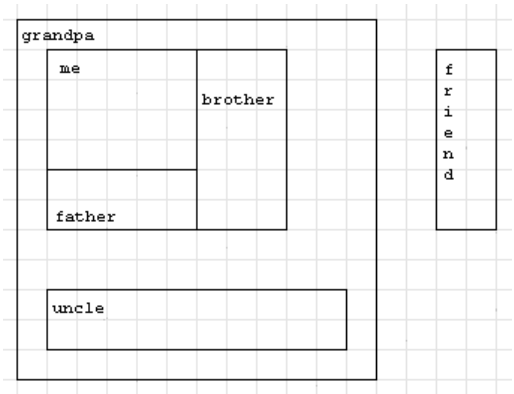
```
15
Widget pack(10,10)
HBox dummy
HBox x
VBox y
y.pack(dummy)
y.set_border(5)
y.set_spacing(55)
dummy.set_border(10)
dummy.set_spacing(20)
x.set_border(10)
x.set_spacing(10)
x.pack(pack)
x.pack(dummy)
x.pack(pack)
x.set_border(0)
```

output

```
dummy 0 0
pack 10 10
x 40 10
y 10 10
```

Note

In the first sample the widgets are arranged as follows:



E. Chip Play

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Let's consider the following game. We have a rectangular field $n \times m$ in size. Some squares of the field contain chips.

Each chip has an arrow painted on it. Thus, each chip on the field points in one of the following directions: up, down, left or right.

The player may choose a chip and make a move with it.

The move is the following sequence of actions. The chosen chip is marked as the current one. After that the player checks whether there are more chips in the same row (or in the same column) with the current one that are pointed by the arrow on the current chip. If there is at least one chip then the closest of them is marked as the new current chip and the former current chip is removed from the field. After that the check is repeated. This process can be repeated several times. If a new chip is not found, then the current chip is removed from the field and the player's move ends.

By the end of a move the player receives several points equal to the number of the deleted chips.

By the given initial chip arrangement determine the maximum number of points that a player can receive during one move. Also determine the number of such moves.

Input

The first line contains two integers n and m ($1 \le n, m, n \times m \le 5000$). Then follow n lines containing m characters each — that is the game field description. "`.`" means that this square is empty. "`L`", "`R`", "`U`", "`D`" mean that this square contains a chip and an arrow on it says left, right, up or down correspondingly.

It is guaranteed that a field has at least one chip.

Output

Print two numbers — the maximal number of points a player can get after a move and the number of moves that allow receiving this maximum number of points.

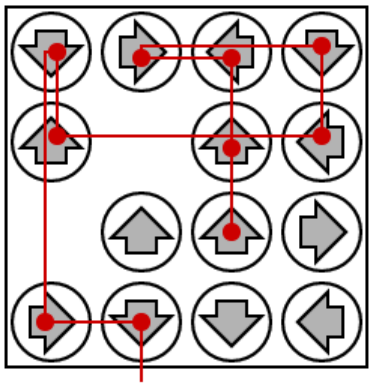
Sample test(s)

input
4 4 DRLD U.UL .UUR RDDL
output
10 1

input
3 5 .D... RRRLL .U...
output
6 2

Note

In the first sample the maximum number of points is earned by the chip in the position $(3, 3)$. You can see its progress at the following picture:



All other chips earn fewer points.