# Codeforces Beta Round #74 (Div. 1 Only)

## A. 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<br>2 3 |

| output |
| --- |
| 0 |

| input |
| --- |
| 3 2 2<br>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.

# B. Widget Library

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

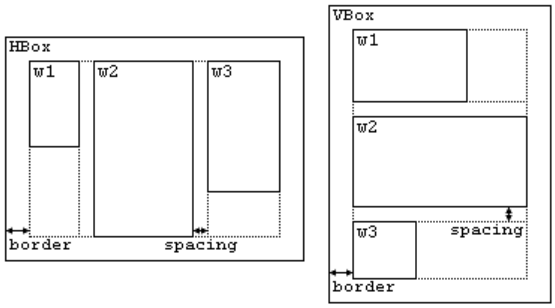
Vasya writes his own library for building graphical user interface. Vasya called his creation `VTK` (`VasyaToolKit`). One of the interesting aspects of this library is that widgets are packed in each other.

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 $0 \times 0$, 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

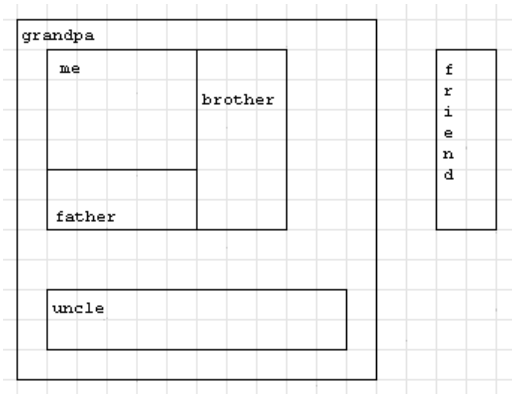
```
15
Widget pack(10,10)
HBox dummy
HBox x
VBox y
y.pack(dummy)
y.set_border(5)
y.set_spacing(55)
dummy.set_border(10)
dummy.set_spacing(20)
x.set_border(10)
x.set_spacing(10)
x.pack(pack)
x.pack(dummy)
x.pack(pack)
x.set_border(0)
```

output

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

## Note

In the first sample the widgets are arranged as follows:



# C. 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.

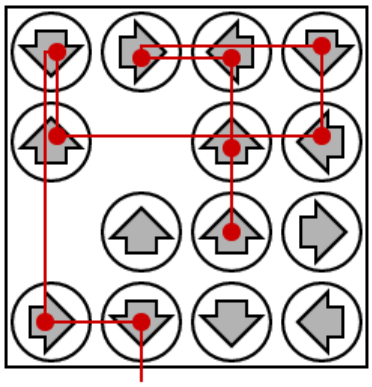
**Sample test(s)**

| input |
| --- |
| 4 4<br>DRLD<br>U.UL<br>.UUR<br>RDDL |
| output |
| 10 1 |

| input |
| --- |
| 3 5<br>.D...<br>RRRLL<br>.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.

# D. Space mines

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

Once upon a time in the galaxy of far, far away...

Darth Wader found out the location of a rebels' base. Now he is going to destroy the base (and the whole planet that the base is located at), using the Death Star.

When the rebels learnt that the Death Star was coming, they decided to use their new secret weapon — space mines. Let's describe a space mine's build.

Each space mine is shaped like a ball (we'll call it the mine body) of a certain radius $r$ with the center in the point $O$. Several spikes protrude from the center. Each spike can be represented as a segment, connecting the center of the mine with some point $P$, such that $r < |OP| \le \frac{3}{2}r$ (transporting long-spiked mines is problematic), where $|OP|$ is the length of the segment connecting $O$ and $P$. It is convenient to describe the point $P$ by a vector $p$ such that $P = O + p$.

The Death Star is shaped like a ball with the radius of $R$ ($R$ exceeds any mine's radius). It moves at a constant speed along the $v$ vector at the speed equal to $|v|$. At the moment the rebels noticed the Star of Death, it was located in the point $A$.

The rebels located $n$ space mines along the Death Star's way. You may regard the mines as being idle. The Death Star does not know about the mines' existence and cannot notice them, which is why it doesn't change the direction of its movement. As soon as the Star of Death touched the mine (its body or one of the spikes), the mine bursts and destroys the Star of Death. A touching is the situation when there is a point in space which belongs both to the mine and to the Death Star. It is considered that Death Star will not be destroyed if it can move infinitely long time without touching the mines.

Help the rebels determine whether they will succeed in destroying the Death Star using space mines or not. If they will succeed, determine the moment of time when it will happen (starting from the moment the Death Star was noticed).

## Input

The first input data line contains $7$ integers $A_x, A_y, A_z, v_x, v_y, v_z, R$. They are the Death Star's initial position, the direction of its movement, and its radius ( $-10 \le v_x, v_y, v_z \le 10$, $|v| > 0$, $0 < R \le 100$).

The second line contains an integer $n$, which is the number of mines ($1 \le n \le 100$). Then follow $n$ data blocks, the $i$-th of them describes the $i$-th mine.

The first line of each block contains $5$ integers $O_{ix}, O_{iy}, O_{iz}, r_i, m_i$, which are the coordinates of the mine centre, the radius of its body and the number of spikes ($0 < r_i < 100$, $0 \le m_i \le 10$). Then follow $m_i$ lines, describing the spikes of the $i$-th mine, where the $j$-th of them describes the $i$-th spike and contains $3$ integers $p_{ijx}, p_{ijy}, p_{ijz}$ — the coordinates of the vector where the given spike is directed ($r_i < |p_{ij}| \le \frac{3}{2}r_i$).

The coordinates of the mines' centers and the center of the Death Star are integers, their absolute value does not exceed $10000$. It is guaranteed that $R > r_i$ for any $1 \le i \le n$. For any mines $i \ne j$ the following inequality if fulfilled: $|O_iO_j| > \frac{3}{2}(r_i + r_j)$. Initially the Death Star and the mines do not have common points.

## Output

If the rebels will succeed in stopping the Death Star using space mines, print the time from the moment the Death Star was noticed to the blast.

If the Death Star will not touch a mine, print "-1" (without quotes).

For the answer the absolute or relative error of $10^{-6}$ is acceptable.

**Sample test(s)**

input

```
0 0 0 1 0 0 5
2
10 8 0 2 2
0 -3 0
2 2 0
20 0 0 4 3
2 4 0
-4 3 0
1 -5 0
```

output

```
10.0000000000
```

input

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

output

```
-1
```

input

```
30 30 2 1 2 1 20
3
0 0 40 5 1
1 4 4
-10 -40 -5 7 0
100 200 95 8 1
-10 0 0
```

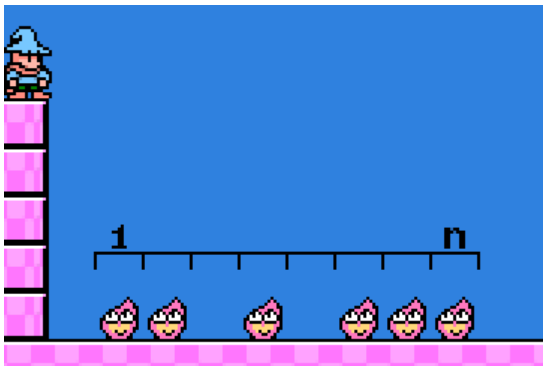
output

```
74.6757620881
```

# E. Fire and Ice

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The Fire Lord attacked the Frost Kingdom. He has already got to the Ice Fortress, where the Snow Queen dwells. He arranged his army on a segment $n$ in length not far from the city walls. And only the frost magician Solomon can save the Frost Kingdom.



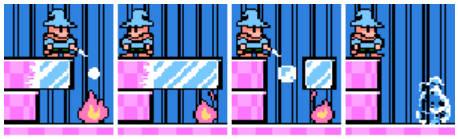

The $n$-long segment is located at a distance equal exactly to $1$ from the castle walls. It can be imaginarily divided into unit segments. On some of the unit segments *fire demons* are located — no more than one demon per position. Each demon is characterised by his *strength* - by some positive integer. We can regard the fire demons being idle.

Initially Solomon is positioned on the fortress wall. He can perform the following actions several times in a row:

- "L" — Solomon shifts one unit to the left. This movement cannot be performed on the castle wall.
- "R" — Solomon shifts one unit to the left. This movement cannot be performed if there's no ice block to the right.
- "A" — If there's nothing to the right of Solomon, then Solomon creates an ice block that immediately freezes to the block that Solomon is currently standing on. If there already is an ice block, then Solomon destroys it. At that the ice blocks to the right of the destroyed one can remain but they are left unsupported. Those ice blocks fall down.

Solomon spends exactly a second on each of these actions.

As the result of Solomon's actions, ice blocks' segments fall down. When an ice block falls on a fire demon, the block evaporates and the demon's strength is reduced by $1$. When the demons' strength is equal to $0$, the fire demon vanishes. The picture below shows how it happens. The ice block that falls on the position with no demon, breaks into lots of tiny pieces and vanishes without hurting anybody.



Help Solomon destroy all the Fire Lord's army in minimum time.

## Input

The first line contains an integer $n$ ($1 \le n \le 1000$). The next line contains $n$ numbers, the $i$-th of them represents the strength of the fire demon standing of the $i$-th position, an integer from $1$ to $100$. If there's no demon on the $i$-th position, then the $i$-th number equals to $0$. It is guaranteed that the input data have at least one fire demon.

## Output

Print a string of minimum length, containing characters "L", "R" and "A" — the succession of actions leading to the required result.

If there are several possible answers, print any of them.

**Sample test(s)**

| input |
| --- |
| 3<br>1 0 1 |
| output |
| ARARARALLLA |

| input |
| --- |
| 3<br>0 2 0 |
| output |
| ARARALAARALA |