A. Question Marks

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   256 megabytes
```

Tim is doing a test consisting of $4n$ questions; each question has 4 options: 'A', 'B', 'C', and 'D'. For each option, there are exactly n correct answers corresponding to that option — meaning there are n questions with the answer 'A', n questions with the answer 'B', n questions with the answer 'C', and n questions with the answer 'D'.

For each question, Tim wrote his answer on the answer sheet. If he could not figure out the answer, he would leave a question mark '?' for that question.

You are given his answer sheet of $4n$ characters. What is the maximum number of correct answers Tim can get?

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains an integer n ($1 \le n \le 100$).

The second line of each test case contains a string s of $4n$ characters ($s_i \in \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}, \mathtt{?}\}$) — Tim's answers for the questions.

Output

For each test case, print a single integer — the maximum score that Tim can achieve.

Standard Input	Standard Output
6 1 ABCD 2 AAAAAAAA 2 AAAABBBB 2 ???????? 3 ABCABCABCABC 5 ACADC??ACAC?DCAABC?C	4 2 4 0 9 13

Note

In the first test case, there is exactly one question with each answer 'A', 'B', 'C', and 'D'; so it's possible that Tim gets all his answers correct.

In the second test case, there are only two correct answers 'A' which makes him get exactly 2 points in any case.

In the third test case, Tim can get at most 2 correct answers with option 'A' and 2 correct answers with option 'B'. For example, he would get 4 points if the answers were '`AACCBBDD`'.

In the fourth test case, he refuses to answer any question at all, which makes him get 0 points.

B. Parity and Sum

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   256 megabytes
```

Given an array a of n positive integers.

In one operation, you can pick any pair of indexes (i, j) such that a_i and a_j have **distinct** parity, then replace the smaller one with the sum of them. More formally:

- If $a_i < a_j$, replace a_i with $a_i + a_j$;
- Otherwise, replace a_j with $a_i + a_j$.

Find the minimum number of operations needed to make all elements of the array have the same parity.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the elements of array a.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer — the minimum number of operations required.

Standard Input	Standard Output
7	0
5	0
1 3 5 7 9	2
4	4
4 4 4 4	3
3	3
2 3 4	3
4	
3 2 2 8	
6	
4 3 6 1 2 1	
6	
3 6 1 2 1 2	
5	
999999996 999999997 999999998 999999999 1000000000	

Note

In the first test case, all integers already have the same parity. Therefore, no operation is needed.

In the third test case, we can perform two operations $(1, 2)$ and $(1, 3)$. The array a transforms as follows: $a = [2, 3, 4] \longrightarrow [5, 3, 4] \longrightarrow [5, 3, 9]$.

In the fourth test case, an example of an optimal sequence of operations is $(1, 2)$, $(1, 3)$, $(1, 4)$, and $(1, 4)$. The array a transforms as follows:
$a = [3, 2, 2, 8] \longrightarrow [3, 5, 2, 8] \longrightarrow [3, 5, 5, 8] \longrightarrow [11, 5, 5, 8] \longrightarrow [11, 5, 5, 19]$.

C. Light Switches

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

There is an apartment consisting of n rooms, each with its light **initially turned off**.

To control the lights in these rooms, the owner of the apartment decided to install chips in the rooms so that each room has exactly one chip, and the chips are installed at different times. Specifically, these times are represented by the array $a_1, a_2, \ldots, a_n$, where a_i is the time (in minutes) at which a chip is installed in the i-th room.

As soon as a chip is installed, it changes the room's light status every k minutes — it turns on the light for k minutes, then turns it off for the next k minutes, then turns it back on for the next k minutes, and so on. In other words, the light status is changed by the chip at minute a_i, $a_i + k$, $a_i + 2k$, $a_i + 3k$, $\ldots$ for the i-th room.

What is the earliest moment when all rooms in the apartment have their lights turned on?

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the number of rooms in the apartment and the period of the chips.

The second line contains n **distinct** integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the moments when the chips are installed.

It is guaranteed that the sum of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, print a single integer — the answer to the question (in minutes). If there is no such moment that the lights are turned on in all the rooms, print -1 instead.

Standard Input	Standard Output

9
4 4
2 3 4 5
4 3
2 3 4 5
4 3
3 4 8 9
3 3
6 2 1
1 1
1
7 5
14 34 6 25 46 7 17
6 5
40 80 99 60 90 50
6 5
	5
	-1
	10
	8
	1
	47
	100
	-1
	-1

64 40 50 68 70 10
2 1
1 1000000000	

Note

In the first test case, all lights will be on by the minute 5 without any of them being turned off by the chips. The answer is 5.

In the second test case, due to $k = 3$, the 1-st light will be on at minutes $2, 3, 4, 8, 9, 10, 14, \ldots$; meanwhile, the 4-th light will be on at minutes $5, 6, 7, 11, 12, 13, 17, \ldots$. These two sequences don't have any number in common, so they will never be on at the same time.

In the third test case, it can be seen that the 1-st and 2-nd lights will be turned off at minutes 6 and 7, but the chips will turn them back on at minutes 9 and 10. The 3-rd and 4-th lights will also be on at minute 10, so the answer is 10.

D. Med-imize

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

Given two positive integers n and k, and another array a of n integers.

In one operation, you can select any subarray of size k of a, then remove it from the array without changing the order of other elements. More formally, let (l, r) be an operation on subarray $a_l, a_{l+1}, \ldots, a_r$ such that $r - l + 1 = k$, then performing this operation means replacing a with $[a_1, \ldots, a_{l-1}, a_{r+1}, \ldots, a_n]$.

For example, if $a = [1, 2, 3, 4, 5]$ and we perform operation $(3, 5)$ on this array, it will become $a = [1, 2]$. Moreover, operation $(2, 4)$ results in $a = [1, 5]$, and operation $(1, 3)$ results in $a = [4, 5]$.

You have to repeat the operation while the length of a is greater than k (which means $|a| > k$). What is the largest possible median[†] of all remaining elements of the array a after the process?

[†]The median of an array of length n is the element whose index is $\lfloor (n+1)/2 \rfloor$ after we sort the elements in non-decreasing order. For example: $median([2, 1, 5, 4, 3]) = 3$, $median([5]) = 5$, and $median([6, 8, 2, 4]) = 4$.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains two integers n and k ($1 \le n, k \le 5 \cdot 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the array a.

It is guaranteed that the sum of n over all test cases does not exceed $5 \cdot 10^5$.

Output

For each test case, print a single integer — the largest median possible after performing the operations.

Standard Input	Standard Output
5	3
4 3	4
3 9 9 2	9
5 3	6
3 2 5 6 4	4
7 1	
5 9 2 6 5 4 6	
8 2	
7 1 2 6 8 3 4 5	
4 5	
3 4 5 6	

Note

In the first test case, you can select a subarray (l, r) which can be either $(1, 3)$ or $(2, 4)$. Thus, two obtainable final arrays are $[3]$ and $[2]$. The former one has the larger median ($3 > 2$) so the answer is 3.

In the second test case, three obtainable final arrays are $[6, 4]$, $[3, 4]$, and $[3, 2]$. Their medians are 4, 3, and 2 respectively. The answer is 4.

In the third test case, only one element is left in the final array and it can be any element of the initial array. The largest one among them is 9, so the answer is 9.

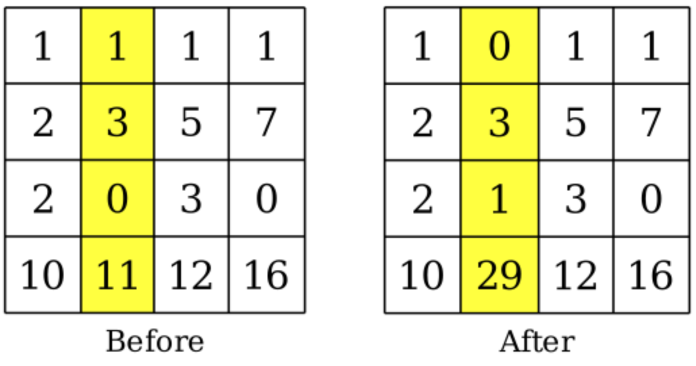
E. Xor-Grid Problem

```
Input file:     standard input
Output file:    standard output
Time limit:     5 seconds
Memory limit:   256 megabytes
```

Given a matrix a of size $n \times m$, each cell of which contains a non-negative integer. The integer lying at the intersection of the i-th row and the j-th column of the matrix is called $a_{i,j}$.

Let's define $f(i)$ and $g(j)$ as the XOR of all integers in the i-th row and the j-th column, respectively. In one operation, you can either:

- Select any row i, then assign $a_{i,j} := g(j)$ for each $1 \le j \le m$; or
- Select any column j, then assign $a_{i,j} := f(i)$ for each $1 \le i \le n$.



An example of applying an operation on column 2 of the matrix.

In this example, as we apply an operation on column 2, all elements in this column are changed:

- $a_{1,2} := f(1) = a_{1,1} \oplus a_{1,2} \oplus a_{1,3} \oplus a_{1,4} = 1 \oplus 1 \oplus 1 \oplus 1 = 0$
- $a_{2,2} := f(2) = a_{2,1} \oplus a_{2,2} \oplus a_{2,3} \oplus a_{2,4} = 2 \oplus 3 \oplus 5 \oplus 7 = 3$
- $a_{3,2} := f(3) = a_{3,1} \oplus a_{3,2} \oplus a_{3,3} \oplus a_{3,4} = 2 \oplus 0 \oplus 3 \oplus 0 = 1$
- $a_{4,2} := f(4) = a_{4,1} \oplus a_{4,2} \oplus a_{4,3} \oplus a_{4,4} = 10 \oplus 11 \oplus 12 \oplus 16 = 29$

You can apply the operations any number of times. Then, we calculate the $beauty$ of the final matrix by summing the absolute differences between all pairs of its adjacent cells.

More formally, $beauty(a) = \sum |a_{x,y} - a_{r,c}|$ for all cells (x, y) and (r, c) if they are adjacent. Two cells are considered adjacent if they share a side.

Find the minimum $beauty$ among all obtainable matrices.

Input

The first line contains a single integer t $(1 \le t \le 250)$ — the number of test cases.

The first line of each test case contains two integers n and m $(1 \le n, m \le 15)$ — the number of rows and columns of a, respectively.

The next n lines, each containing m integers $a_{i,1}, a_{i,2}, \ldots, a_{i,m}$ $(0 \le a_{i,j} < 2^{20})$ — description of the matrix a.

It is guaranteed that the sum of $(n^2 + m^2)$ over all test cases does not exceed 500.

Output

For each test case, print a single integer b — the smallest possible $beauty$ of the matrix.

Standard Input

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

Standard Output

```
1
3
13
24
```

Note

Let's denote $r(i)$ as the first type operation applied on the i-th row, and $c(j)$ as the second type operation applied on the j-th column.

In the first test case, you can apply an operation $c(1)$, which assigns $a_{1,1} := 1 \oplus 3 = 2$. Then, we'll receive this matrix:

2	3

In the second test case, you can apply an operation $r(1)$, which assigns:

- $a_{1,1} := g(1) = 0 \oplus 5 = 5$
- $a_{1,2} := g(2) = 1 \oplus 4 = 5$
- $a_{1,3} := g(3) = 0 \oplus 4 = 4$

The resulting matrix after performing the operation is:

5	5	4
5	4	4

In the third test case, the best way to achieve minimum $beauty$ is applying three operations: $c(3)$, $r(2)$, and $c(2)$. The resulting matrix is:

0	4	6
4	5	6

,

F1. Dyn-scripted Robot (Easy Version)

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

This is the easy version of the problem. The only difference is that in this version $k \le n$. You can make hacks only if both versions of the problem are solved.

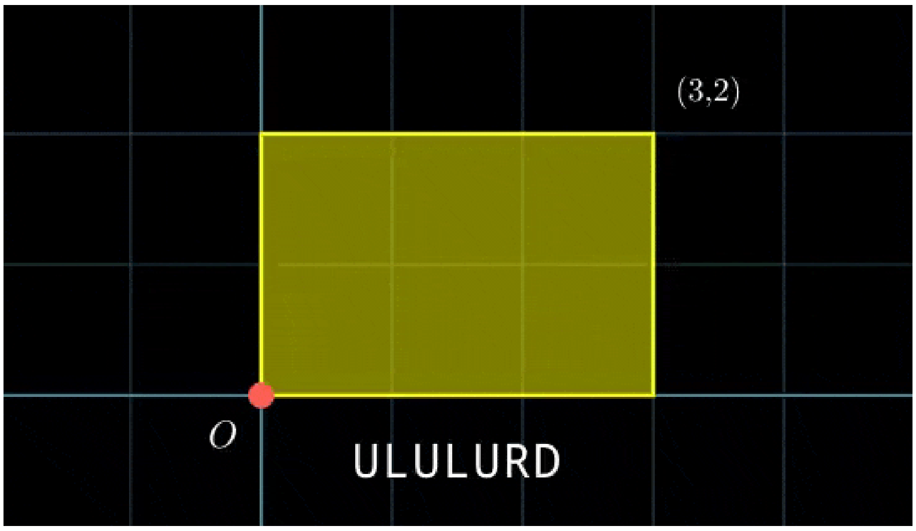
Given a $w \times h$ rectangle on the Oxy plane, with points $(0,0)$ at the bottom-left and (w,h) at the top-right of the rectangle.

You also have a robot initially at point $(0,0)$ and a script s of n characters. Each character is either L, R, U, or D, which tells the robot to move left, right, up, or down respectively.

The robot can only move inside the rectangle; otherwise, it will change the script s as follows:

- If it tries to move outside a vertical border, it changes all L characters to R's (and vice versa, all R's to L's).
- If it tries to move outside a horizontal border, it changes all U characters to D's (and vice versa, all D's to U's).

Then, it will execute the changed script starting from the character which it couldn't execute.



An example of the robot's movement process, $s = \texttt{"ULULURD"}$

The script s will be executed for k times continuously. All changes to the string s will be retained even when it is repeated. During this process, how many times will the robot move to the point $(0,0)$ in total? **Note that the initial position does NOT count**.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains four integers n, k, w, and h ($1 \le n, w, h \le 10^6$; $1 \le k \le n$).

The second line contains a single string s of size n ($s_i \in \{\texttt{L}, \texttt{R}, \texttt{U}, \texttt{D}\}$) — the script to be executed.

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

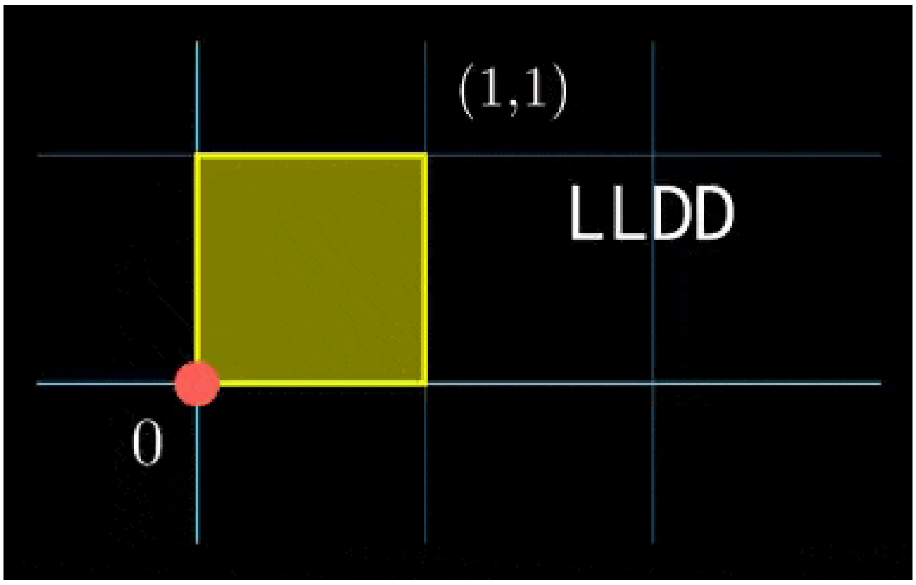
For each test case, print a single integer — the number of times the robot reaches $(0,0)$ when executing script s for k times continuously.

Standard Input	Standard Output
5 2 2 2 2 UR 4 2 1 1 LLDD 6 3 3 1 RLRRRL 5 5 3 3 RUURD 7 5 3 4 RRDLUUU	0 4 3 0 1

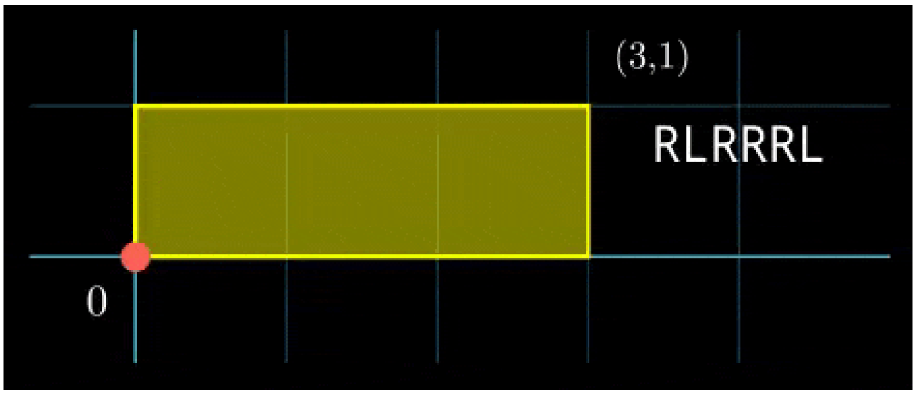
Note

In the first test case, the robot only moves up and right. In the end, it occupies the position $(2,2)$ but never visits $(0,0)$. So the answer is 0.

In the second test case, each time executing the script the robot visits the origin twice. And since $k=2$, it visits the origin $2 \cdot 2 = 4$ times overall.



In the third test case, the visualization is shown as below:



,

F2. Dyn-scripted Robot (Hard Version)

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

This is the hard version of the problem. The only difference is that in this version $k \le 10^{12}$. You can make hacks only if both versions of the problem are solved.

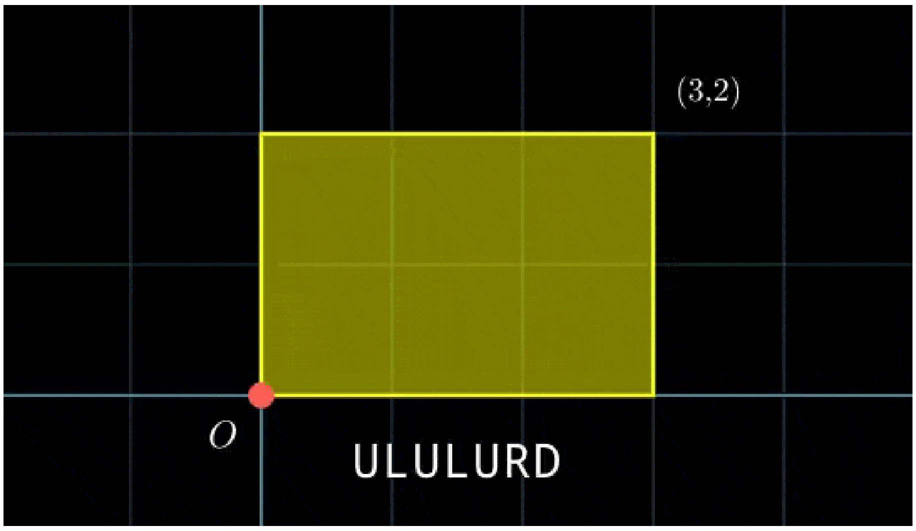
Given a $w \times h$ rectangle on the Oxy plane, with points $(0,0)$ at the bottom-left and (w,h) at the top-right of the rectangle.

You also have a robot initially at point $(0,0)$ and a script s of n characters. Each character is either L, R, U, or D, which tells the robot to move left, right, up, or down respectively.

The robot can only move inside the rectangle; otherwise, it will change the script s as follows:

- If it tries to move outside a vertical border, it changes all L characters to R's (and vice versa, all R's to L's).
- If it tries to move outside a horizontal border, it changes all U characters to D's (and vice versa, all D's to U's).

Then, it will execute the changed script starting from the character which it couldn't execute.



An example of the robot's movement process, $s = \texttt{"ULULURD"}$

The script s will be executed for k times continuously. All changes to the string s will be retained even when it is repeated. During this process, how many times will the robot move to the point $(0,0)$ in total? **Note that the initial position does NOT count**.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases.

The first line of each test case contains four integers n, k, w, and h ($1 \le n, w, h \le 10^6$; $1 \le k \le 10^{12}$).

The second line contains a single string s of size n ($s_i \in \{\texttt{L}, \texttt{R}, \texttt{U}, \texttt{D}\}$) — the script to be executed.

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case, print a single integer — the number of times the robot reaches $(0, 0)$ when executing script s for k times continuously.

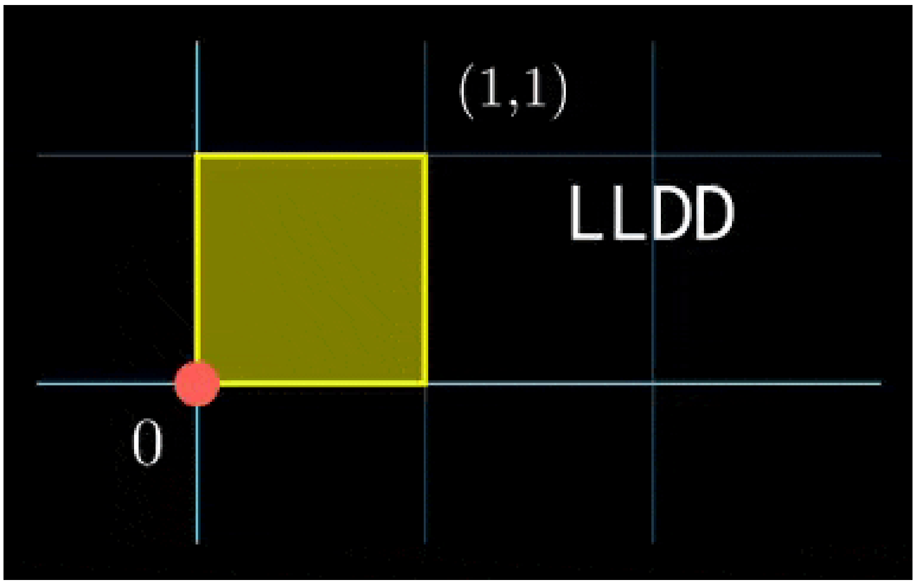
Standard Input	Standard Output

`6`
`2 4 2 2`
`UR`
`4 2 1 1`
`LLDD`
`6 3 3 1`
`RLRRRL`
`5 6 3 3`
`RUURD`
`7 5 3 4`
`RRDLUUU`
`7 123456789999 3 2`
`ULULURD`
	`1`
	`4`
	`3`
	`1`
	`1`
	`41152263332`

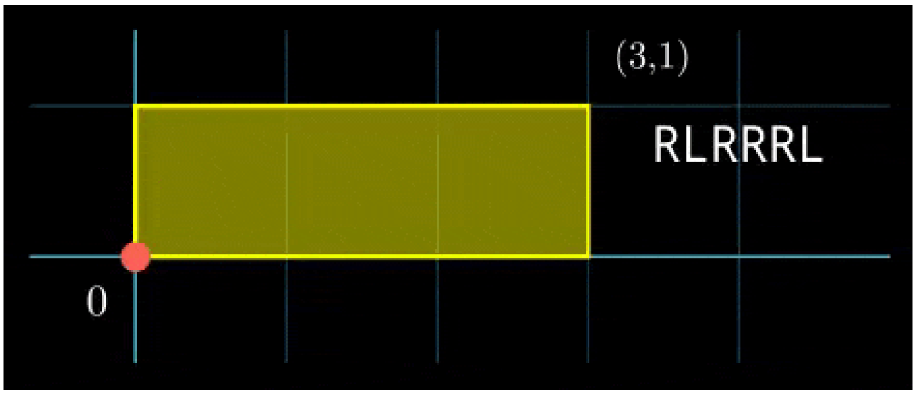
Note

In the first test case, the robot only moves up and right for the first two executions. After that, it occupies the position $(2, 2)$. For the next two executions, it moves down and left and finishes at $(0, 0)$. So the answer is 1.

In the second test case, each time executing the script the robot visits the origin twice. And since $k = 2$, it visits the origin $2 \cdot 2 = 4$ times overall.



In the third test case, the visualization is shown as below:


(3,1)
RLRRRL
0