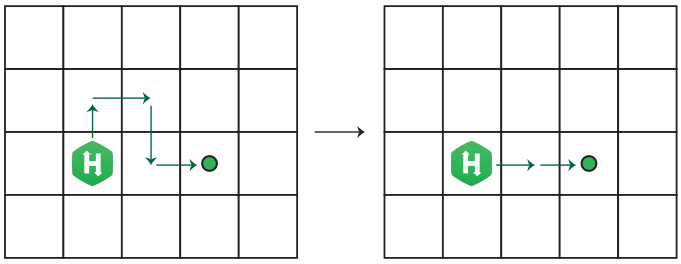
Question - 1
Character Reprogramming

A control device has 4 buttons that can be used to move a character around a screen in 4 directions: Up (U), Down (D), Left (L), and Right (R). The movement needs to be optimized by deleting unnecessary instructions while maintaining the same destination. Given the original set of instructions, what is the maximum number that can be deleted and still have the character reach the destination?

Note: The instructions that are deleted do not need to be contiguous.

Example
s = 'URDR'



Given an original set of instructions *s = 'URDR'*, the final destination is 2 units to the right of the initial position after the character moves up, right, down, and right. If '*U*' and '*D*' are deleted, the destination remains the same. The answer 2 will be returned.

Function Description
Complete the function *getMaxDeletions* in the editor below.

getMaxDeletions has the following parameter:
string *s:* the original instructions that were programmed

Returns:
int: the maximum number of instructions that can be deleted from *s* while maintaining the destination

Constraints

- $1 \le n \le 10^5$
- *s* contains only the characters 'U', 'D', 'L', and 'R'.

▼ Input Format For Custom Testing

The first line contains a string, *s*, denoting the instructions.

▼ Sample Case 0

/

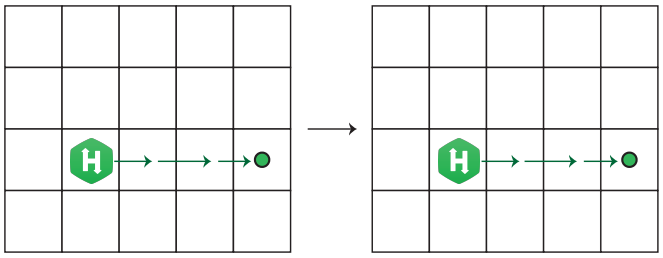
Sample Input

```
STDIN       Function
-----       -----
RRR     →   s = 'RRR'
```

Sample Output

```
0
```

Explanation



"R R R" "R R R"

There is nothing that can be deleted from these instructions, so the answer is 0.

▼ Sample Case 1

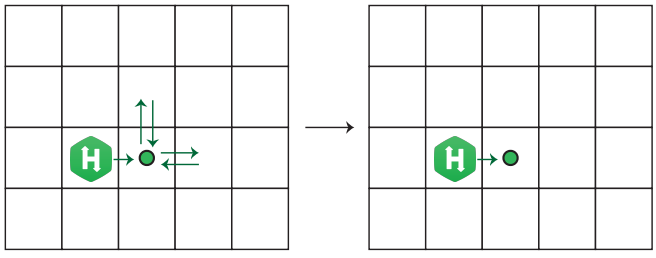
Sample Input

```
STDIN       Function
-----       -----
RUDRL   →   s = 'RUDRL'
```

Sample Output

```
4
```

Explanation



"R U D R L" "R Ø Ø R L"

The final destination is 1 unit to the right of the initial position. Deleting the final 2 instructions, '*RL*', will not affect the destination. Deleting *'UD'* also will not affect the destination. Because the goal is to delete the maximum number of movements, both of these instances, '*UDRL*', should be deleted, which is a total of 4 deletions.

Question - 2
Conference Schedule

A schedule has just been released for an upcoming tech conference. The schedule provides the start and end times of each of the presentations. Once a presentation has begun, no one can enter or

/

leave the room. It takes no time to go from one presentation to another. Determine the maximum number of presentations that can be attended by one person.

Example

n = 3

scheduleStart = [1, 1, 2]

scheduleEnd = [3, 2, 4]

Using 0-based indexing, an attendee could attend any presentation alone, or both presentations *1* and *2*. Presentation *0* ends too late to be able to attend presentation *2* afterwards. The maximum number of presentations one person can attend is *2*.

Function Description

Complete the function *maxPresentations* in the editor below.

maxPresentations has the following parameter(s):

scheduleStart[n]: the start times of presentation *i*

scheduleEnd[n]: the end times of presentation *i*

Returns:

int: the maximum number of presentations that can be attended by one person

Constraints

- $1 \le n \le 10^5$
- $1 \le scheduleStart[i], scheduleEnd[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *scheduleStart[]*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *scheduleStart[i]*.

The next line contains an integer, *n*, the number of elements in *scheduleEnd[]*.

Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains an integer that describes *scheduleEnd[i]*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       -----
4       →   scheduleStart[] size n = 4
1       →   scheduleStart = [1, 1, 2, 3]
1
2
3
4       →   scheduleEnd[] size n = 4
2       →   scheduleEnd = [2, 3, 3, 4]
3
3
4
```

Sample Output

```
3
```

Explanation

/

An attendee can go to presentations *0, 2*, and *3* without conflict. If presentation *1* is chosen, from time *1* to *3*, only two presentations can be attended. The maximum number of presentations a single person can attend is *3*.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       -----
4       →   scheduleStart[] size n = 4
6       →   scheduleStart = [6, 1, 2, 3]
1
2
4
4       →   scheduleEnd[] size n = 4
8       →   scheduleEnd = [8, 9, 4, 7]
9
4
7
```

Sample Output

```
2
```

Explanation
An attendee can attend presentation *1* only as it runs the entire day, or they can instead attend meeting *2* from *2* until *4*, then choose to attend either presentation *0* or *3*. The maximum number of presentations a single person can attend is *2*.

Question - 3
Grouping Transactions by Items' Names

For a given array of transactions, group all of the transactions by item name. Return an array of strings where each string contains the item name followed by a space and the number of associated transactions.

Note: Sort the array descending by transaction count, then ascending alphabetically by item name for items with matching transaction counts.

Example
transactions = ['notebook', 'notebook', 'mouse', 'keyboard', 'mouse']

There are two items with *2* transactions each: *'notebook'* and *'mouse'*. In alphabetical order, they are *'mouse', 'notebook'*.
There is one item with *1* transaction: '*keyboard*'.
The return array, sorted as required, is *['mouse 2', 'notebook 2', 'keyboard 1']*.

Function Description
Complete the function *groupTransactions* in the editor below.

groupTransactions has the following parameter(s):
string transactions[n]: each *transactions[i]* denotes the item name in the i^{th} transaction

/

Returns:

string[]: an array of strings of "item name[space]transaction count" sorted as described

Constraints

- $1 \le n \le 10^5$
- 1 ≤ length of *transactions[i]* ≤ 10
- *transactions[i]* contains only lowercase English letters, ascii[a-z]

▼ Input Format Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains a single integer, *n*, the size of *transactions*.
Each of the next *n* lines contains a string, the item name for *transactions[i]*.

▼ Sample Case 0

Sample Input

```
STDIN     Function
-----     -----
4      →  transactions[] size n = 4
bin    →  transactions = ['bin', 'can', 'bin',
'bin']
can
bin
bin
```

Sample Output

```
bin 3
can 1
```

Explanation

- There is one item '*bin*' with *3* transactions.
- There is one item '*can*' with *1* transaction.
- The return array sorted descending by transaction count, then ascending by name is *['bin 3', 'can 1']*.

▼ Sample Case 1

Sample Input

```
STDIN     Function
-----     -----
3      →  transactions[] size n = 3
banana →  transactions = ['banana', 'pear',
'apple']
pear
apple
```

Sample Output

```
apple 1
banana 1
pear 1
```

/

Explanation

- There is one item '*apple*' with *1* transaction.
- There is one item *'banana'* with *1* transaction.
- There is one item '*pear*' with *1* transaction.
- The return array sorted descending by transaction count, then ascending by name is *['apple 1', 'banana 1', 'pear 1']*.

/