2_Vim vs Code

(10 分)

時間限制: 1 second

記憶體限制: 256 MB

題目敘述

你都用 Vim 還是 Vscode？

Vim 和 Vscode 之爭，不僅是文字編輯器之間的較量，更是傳統與前衛的對決、經典和新潮的碰撞。於是，程式設計師們形成了兩大教派，Vim 教派和 Vscode 教派，彼此都想證明自己擁護的才是編輯器的正統！

這天，Vscode 教徒有了一個邪惡的計畫。他們認為，Vim 區分一般模式和編輯模式的設計實在是太反人類了，怎麼會有人想用 HJKL 鍵當方向鍵來用？因此他們在 Vim 裡面植入了「Vim Suspiciously Compromised On Direction-keys Entered」插件，簡稱 VSCODE。被植入 VSCODE 插件的 Vim 編輯器會將一些字母誤認為方向鍵，導致使用者無法正常輸入，讓使用者能夠意識到 Vim 的設計多麼不合理。

具體來說，Vim 是一個網格狀的編輯區域，高 H 格寬 W 格、共有 $H \times W$ 格。編輯區域中每一格都可以顯示一個字元，一開始每一格都是空白。此外，編輯時會有一個「游標」決定輸入的位置，一開始在最左上角的格子。當使用者輸入一個字元時，游標所在的格子會填上該字元，且游標往右移動一格；若游標已經在最右邊一行，則游標移動到下一列的最左邊；若游標已經在最右下角，則游標回到最左上角。但因為被植入 VSCODE 插件的關係，編輯時如果輸入了以下四種字元，他們將會被當作方向鍵來處理而不修改編輯區域的內容：

- 輸入小寫字母 `h`，游標向左移動。如果游標已經在最左邊一行，則游標移動到同一列最右邊一行。
- 輸入小寫字母 `j`，游標向下移動。如果游標已經在最下面一列，則游標移動到同一行最上面一列。
- 輸入小寫字母 `k`，游標向上移動。如果游標已經在最上面一列，則游標移動到同一行最下面一列。
- 輸入小寫字母 `l`，游標向右移動。如果游標已經在最右邊一行，則游標移動到同一列最左邊一行。

作為 Vscode 資深教徒的你，擔下了實做 VSCODE 插件的重責大任。現在給你 Vim 編輯區域的高和寬，以及使用者輸入的文字，你能畫出最終編輯區域的內容嗎？

輸入格式

輸入共有兩行。第一行有兩個整數 H、W，分別代表編輯器中編輯區域的高和寬。第二行包含一個由大小寫英文字母組成的字串 s，代表使用者在編輯器中依序輸入的按鍵。

輸出格式

輸出 H 行，每行 W 個字元，代表編輯器版面上最後留下的文字。如果某一格到最後還是空白，請輸出一個點（`.`）。

資料範圍

- $1 \leq H \times W \leq 10^6$
- $1 \leq |s| \leq 5 \times 10^6$
- s 僅包含大小寫英文字母。

測試範例

輸入範例 1

```
3 5
TheQuickBrownFoxJumpsOverTheLazyDog
```

輸出範例 1

```
ereLa
zyDog
mpsOv
```

輸入範例 2

```
3 5
hlkjhkjkljhkljhlkhjlkjhklhlj
```

輸出範例 2

```
.....
.....
.....
```

輸入範例 3

```
3 5
ByTheWayIUseArchwq
```

輸出範例 3

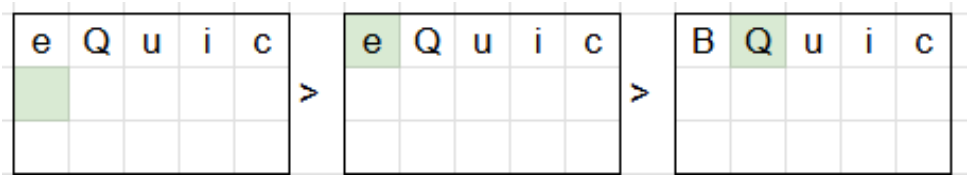
```
ByeWa
yIUse
Arwq.
```

範例說明

在範例一中，輸入到各階段的編輯區內容為：

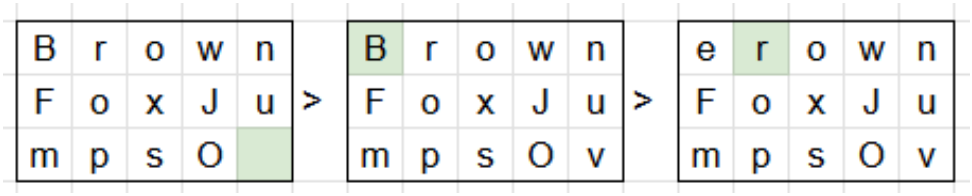
輸入到 `The` 時，這裡的 `h` 使游標向左移動一格，導致接下來輸入的 `e` 覆蓋了原本的 `T`：

輸入到 `TheQuickB`，`k` 使游標上移，使得接下來的 `B` 回到了左上角，覆蓋了原本的 `e`：



輸入到 `TheQuickBrownFoxJu` 時，這裡的 `J` 是大寫的，因此很正常的繼續往下執行。

輸入到 `TheQuickBrownFoxJumpsOve`，輸入 `v` 後來到版面的最後一格，因此回到左上角，使得新增的 `e` 覆蓋了原本的 `B`：



接下來將剩餘的字元處理完畢後就可以得到最後的結果。

在範例二中，輸入的內容全部都是 `hjkl`，僅僅代表了移動而沒有寫入任何資料，因此最後的版面依然為空，並以 `.` 來表示空白的格子。

在範例三中，前後的兩個 `h` 分別導致 `T` 以及 `c` 被覆蓋，其餘的文字皆為正常的輸入。

> `wq` 居然沒有用，究竟要怎麼退出 Vim 呢？

2_Vim vs Code

(10 points)

Time Limit: 1 second

Memory Limit: 256 MB

Statement

Which do you prefer, Vim or VSCode?

The debate between Vim and VSCode is not only a choice between editors, but a battle of tradition and modernity. Programmers have formed two factions: the Vim faction and the VSCode faction, each trying to prove that their editor is superior to the other.

One day, the VSCode followers came up with an evil plan. They believed that it is so absurd that Vim distinguishes between normal mode and insert mode. Who on earth uses HJKL as arrow keys? Therefore, they installed a plugin, Vim Suspiciously Compromised On Direction-keys Entered (abbreviated as the VSCODE plugin), into Vim editors. Vim editors with this plugin installed misinterpret some letters as arrow keys, preventing users from typing normally and making them realize how ridiculous Vim's design is.

To be specific, Vim provides a grid as its editing area. The grid is H cells high and W cells wide, which is $H \times W$ cells in total. Each cell stores a character that may be edited according to the user's input. Additionally, there is a cursor indicating the cell being edited. At the beginning, every cell is initialized with a space character, and the cursor is placed at the top-left corner. When the user enters a key, the cell under the cursor will be overwritten with that key, and the cursor moves one cell to the right; if the cursor is already at the rightmost column, it moves to the leftmost cell in the row below; if the cursor is at the bottom-right corner, it returns to the top-left corner. However, due to the VSCODE plugin, if one of the following four characters is entered, they will be processed like arrow keys without modifying the editing area:

- lowercase `h`: cursor moves left. If the cursor is already at the leftmost column, it moves to the rightmost cell on the same row.
- lowercase `j`: cursor moves down. If the cursor is already at the bottommost row, it moves to the topmost cell on the same column.
- lowercase `k`: cursor moves up. If the cursor is already at the topmost row, it moves to the bottommost cell on the same column.
- lowercase `l`: cursor moves right. If the cursor is already at the rightmost column, it moves to the leftmost cell on the same row.

As a senior VSCode follower, you are in charge of implementing the VSCODE plugin. Given the dimensions of Vim's editing area, along with the user's input text, can you determine the final state of the editing area?

Input Format

A testcase comprises two lines. The first line contains two integers, H and W, representing the height and the width of Vim's editing area. The second line contains a single string s consisting of uppercase and lowercase English letters, which represents the Vim user's input.

Output Format

Output H lines, each containing W characters, representing the characters left on the editing area at the end. For the cells remaining a space character at the end, please print a period (`.`) to indicate them.

Constraints

- $1 \leq H \times W \leq 10^6$
- $1 \leq |s| \leq 5 \times 10^6$
- s comprises uppercase and lowercase English letters only.

Test Cases

Input 1

```
3 5
TheQuickBrownFoxJumpsOverTheLazyDog
```

Output 1

```
ereLa
zyDog
mpsOv
```

Input 2

```
3 5
hlkjhkjkljhkljhlkhjlkjhklhlj
```

Output 2

```
.....
.....
.....
```

Input 3

```
3 5
ByTheWayIUseArchwq
```

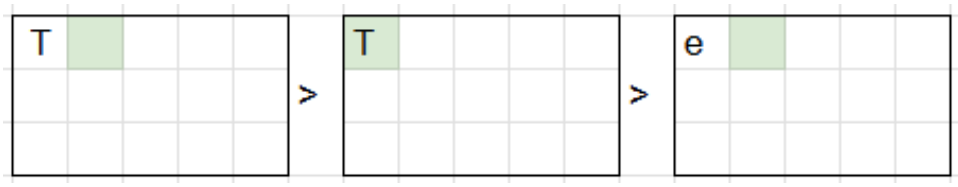
Output 3

```
ByeWa
yIUse
Arwq.
```

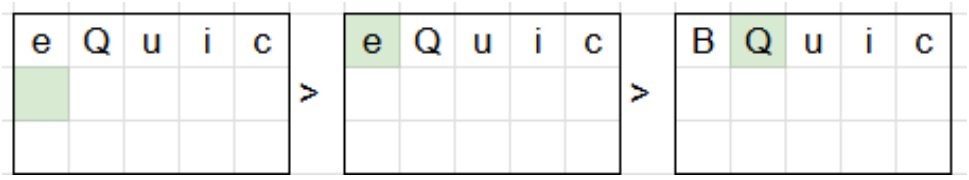
Illustrations

In the example 1,

When we process `The`, the `h` will make the cursor go left instead of putting the `h` into the editor area. The next input, e, will then overwrite the original `T`:

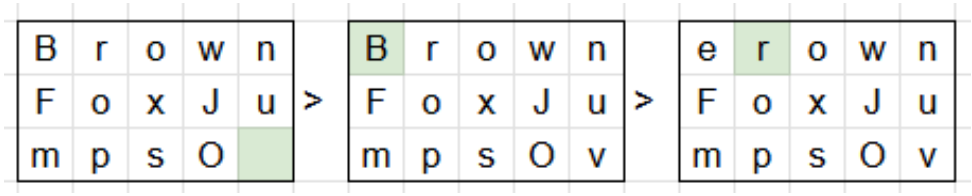


When we process `TheQuickB`, the `k` here will make the cursor go up instead of putting the `k` into the editor area. The next input `B` will then overwrite the original `e`:



When we process to `TheQuickBrownFoxJu`, the `J` is capitalized, so we see it as normal input, putting the `J` into editor area.

When we process to `TheQuickBrownFoxJumpsOve`, the input `v` is at the last slot of the editor area, so the cursor will go back to the first slot of the editor area. The `e` will then overwrite the original `B`:



We can then process the remaining input normally to get the final result.

In example 2, the input is composed of `hjkl`, which are all special characters used to move the cursor. Hence, the final editor area remains empty, and we use `.` to represent the empty slots.

In example 3, the two `h`s in the input will cause `T` and `c` to be overwritten. The remaining input consists of normal character.

> `wq` is not working! How can you exit Vim?