自由之路

> 计算学部十大打卡活动——“龙舞编程新春会”编程打卡（2024-1-29）
>
> 力扣514.自由之路

一、题目

电子游戏“辐射4”中，任务 **“通向自由”** 要求玩家到达名为 **“Freedom Trail Ring”** 的金属表盘，并使用表盘拼写特定关键词才能开门。

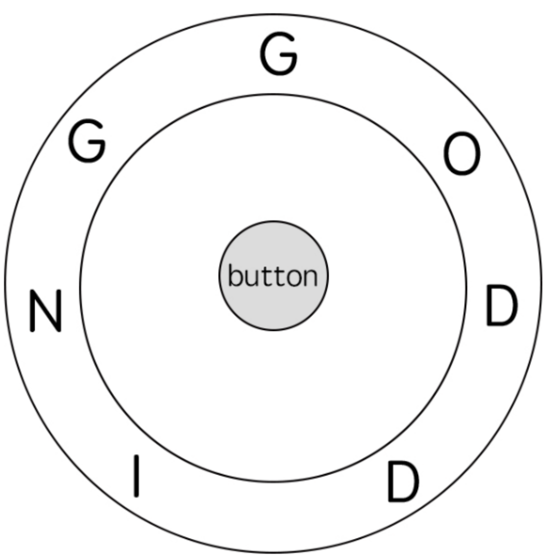
给定一个字符串 `ring` ，表示刻在外环上的编码；给定另一个字符串 `key` ，表示需要拼写的关键词。您需要算出能够拼写关键词中所有字符的**最少**步数。

最初，`ring` 的第一个字符与 `12:00` 方向对齐。您需要顺时针或逆时针旋转 `ring` 以使 `key` 的一个字符在 `12:00` 方向对齐，然后按下中心按钮，以此逐个拼写完 `key` 中的所有字符。

旋转 `ring` 拼出 `key` 字符 `key[i]` 的阶段中：

1. 您可以将 `ring` 顺时针或逆时针旋转 **一个位置** ，计为1步。旋转的最终目的是将字符串 `ring` 的一个字符与 `12:00` 方向对齐，并且这个字符必须等于字符 `key[i]` 。
2. 如果字符 `key[i]` 已经对齐到 `12:00` 方向，您需要按下中心按钮进行拼写，这也将算作 **1 步**。按完之后，您可以开始拼写 `key` 的下一个字符（下一阶段），直至完成所有拼写。

示例 1：



> **输入：** ring = "godding", key = "gd"
>
> **输出：** 4
>
> **解释：**
>
> > 对于 key 的第一个字符 'g'，已经在正确的位置, 我们只需要1步来拼写这个字符。
> > 对于 key 的第二个字符 'd'，我们需要逆时针旋转 ring "godding" 2步使它变成 "ddinggo"。
> > 当然, 我们还需要1步进行拼写。
> > 因此最终的输出是 4。

示例 2：

> **输入：** ring = "godding", key = "godding"
>
> **输出：** 13

提示：

- `1 <= ring.length, key.length <= 100`
- `ring` 和 `key` 只包含小写英文字母
- **保证** 字符串 `key` 一定可以由字符串 `ring` 旋转拼出

二、题解

思路——动态规划

定义 $dp[i][j]$: 表示从前往后拼写出 key 的第 i 个字符， $ring$ 的第 j 个字符与 `12:00` 方向对齐的最少步数（下标均从 0 开始）。

考虑如何进行状态转移，显然，需要知道上一步与 `12:00` 方向对齐的位置，此位置的字符一定等于 key 的第 $i-1$ 个字符。因此需要对每个字符维护一个位置数组 $pos[i]$ ，表示字符 i 在 $ring$ 中出现的位置集合，来加速计算转移的过程。

对于状态 $dp[i][j]$，枚举上一次与 `12:00` 方向对齐的位置 k ，可以列出如下的转移方程：

$$dp[i][j] == \min_{k \in pos[key[i-1]]}\{dp[i-1][k] + \min\{\text{abs}(j-k), n-\text{abs}(j-k)\} + 1\}$$

其中 $\min\{\text{abs}(j-k), n-\text{abs}(j-k)\} + 1$ 表示在当前第 k 个字符与 `12:00` 方向对齐时第 j 个字符旋转到 `12:00` 方向并按下拼写的最少步数。

最后答案即为 $\min_{i=0}^{n-1}\{dp[m-1][i]\}$ 。

代码

```
class Solution {
public:
    int findRotateSteps(string ring, string key) {
        int n = ring.size(), m = key.size();
        vector<int> pos[26];
        vector<vector<int>> dp(m, vector<int>(n, 0x3f3f3f3f));
        int i, j, k;
        // 记录每个字符在ring中出现的位置集合
        for (i = 0; i < n; i++) {
            pos[ring[i] - 'a'].push_back(i);
        }
        //初始化转移矩阵第一行
        for (auto& i : pos[key[0] - 'a']) {
            dp[0][i] = min(i, n - i) + 1;
        }
        for (i = 1; i < m; i++) {
            for (auto& j : pos[key[i] - 'a']) {
                for (auto& k : pos[key[i - 1] - 'a']) {
                    dp[i][j] =
                        min(dp[i][j],
                            dp[i - 1][k] + min(abs(j - k), n - abs(j - k)) + 1);
                }
            }
        }
        return *min_element(dp[m - 1].begin(), dp[m - 1].end()); //获取转移矩阵中最后一行的最小值
    }
};
```

复杂度分析

- 时间复杂度：$O(mn^2)$，其中 m 为字符串 key 的长度，n 为字符串 $ring$ 的长度。一共有 $O(mn)$ 个状态要计算，每次转移的时间复杂度为 $O(n)$，因此时间复杂度为 $O(mn^2)$ 。
- 空间复杂度：$O(mn)$，需要使用 $O(mn)$ 的空间来存放 dp 数组，以及使用 $O(n)$ 的空间来存放 pos 数组，因此总空间复杂度为 $O(mn)$。