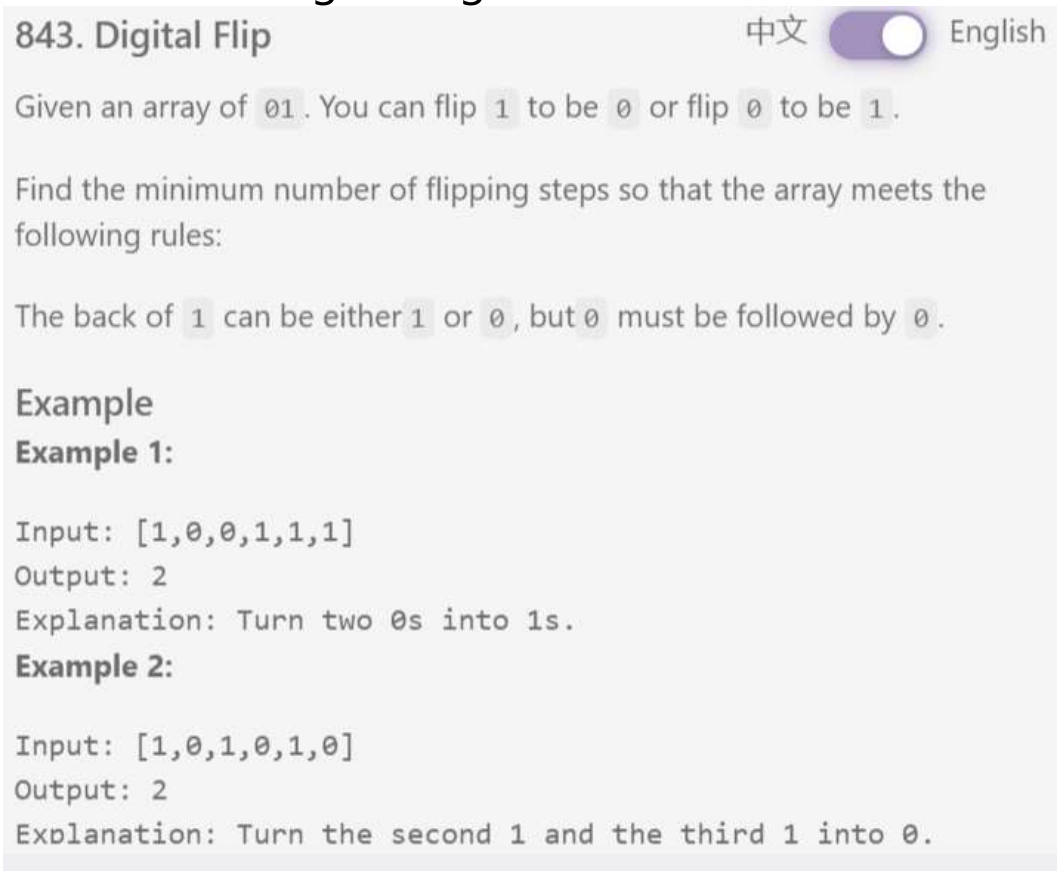
4 序列型动规

笔记本:	DP Note		
创建时间:	10/13/2019 3:16 PM	**更新时间:**	10/23/2019 8:37 PM
作者:	tanziqi1756@outlook.com		

Leetcode152 最大子序列乘积
Leetcode52 最大子序列和
Leetcode256房子涂色付钱问题
Leetcode674 Longest Continuous Increasing Subsequence 最长上升子序列

Lintcode843 Digital Flig



843. Digital Flip

中文 English

Given an array of `01`. You can flip `1` to be `0` or flip `0` to be `1`.

Find the minimum number of flipping steps so that the array meets the following rules:

The back of `1` can be either `1` or `0`, but `0` must be followed by `0`.

Example

Example 1:

```
Input: [1,0,0,1,1,1]
Output: 2
Explanation: Turn two 0s into 1s.
```

Example 2:

```
Input: [1,0,1,0,1,0]
Output: 2
Explanation: Turn the second 1 and the third 1 into 0.
```

直觉：从后往前遍历，遇到第一个1后，把所有0翻转成1.
但是不一定，也可以是：出现第一个1后，把所有1翻转成0.

动态规划组成部分一：确定状态

- 最后一步：最优策略中，最后一位数是否翻转
- 但需要知道前一位数已经变成0还是1
- 并且前N-1位数最少翻转多少次，满足要求（无01子串）
- **不知道的信息加入状态里**
- 状态
- 用f[i][0]表示A[i-1]变成0的情况下，前i位最少翻转多少个能满足要求
- 用f[i][1]表示A[i-1]变成1的情况下，前i位最少翻转多少个能满足要求

动态规划组成部分三：初始条件和边界情况

- 用f[i][0]表示A[i-1]变成0的情况下，前i位最少翻转多少个能满足要求
- 用f[i][1]表示A[i-1]变成1的情况下，前i位最少翻转多少个能满足要求
- $f[i][j] = \min_{(k,j)\neq(0,1)}\{f[i-1][k] + I_{A[i-1]\neq j}\}$
- 初始条件：
  - f[0][0] = f[0][1] = 0

```java
public class Solution {
    /**
     * @param nums: the array
     * @return: the minimum times to flip digit
     */
    public int flipDigit(int[] nums) {
        // Write your code here

        int N = nums.length;
        int[][] dp = new int[N+1][2];

        // initialize
        dp[0][0] = 0;
        dp[0][1] = 0;

        // dp
        // dp[i][0] means nums[i-1] 变成0的情况下，最少需要翻转多少次
        // dp[i][1] means nums[i-1] 变成1的情况下，最少需要翻转多少次
        for( int i = 1; i < dp.length; i++ ) {
            for( int j = 0; j < 2; j++ ) {
                dp[i][j] = Integer.MAX_VALUE;
                // 把A[i-1]变成j,是否需要翻转
                int indicator = (nums[i-1] != j) ? 1 : 0;

                // 把A[i-2]变成k
                for( int k = 0; k < 2; k++ ) {
                    if( k == 0 && j == 1 ) {
                        continue;
                    }
                    dp[i][j] = Math.min(dp[i][j], dp[i-1][k]+indicator);
                }
            }
        }
        return Math.min(dp[N][0], dp[N][1]);
    }
}
```