123-买卖股票的最佳时机

题述

123. 买卖股票的最佳时机 III

难度 困难 1077

给定一个数组，它的第 `i` 个元素是一支给定的股票在第 `i` 天的价格。

设计一个算法来计算你所能获取的最大利润。你最多可以完成 **两笔** 交易。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

示例 1:

输入： prices = [3,3,5,0,0,3,1,4]
输出： 6
解释： 在第 4 天（股票价格 = 0）的时候买入，在第 6 天（股票价格 = 3）的时候卖出，这笔交易所能获得利润 = 3-0 = 3 。
随后，在第 7 天（股票价格 = 1）的时候买入，在第 8 天（股票价格 = 4）的时候卖出，这笔交易所能获得利润 = 4-1 = 3 。

示例 2:

输入： prices = [1,2,3,4,5]
输出： 4
解释： 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。
因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

思路

动态规划

1、确定dp数组以及下标的含义

我们拟定一天有五种状态：

- 0-没有操作
- 1-第一次买入

- 2-第一次卖出
- 3-第二次买入
- 4-第二次卖出

```
dp[i][j] 中i表示第i天，j为0-4五种状态，dp[i][j]表示第i天状态j下所剩余的最大现金
```

2、确定递推公式

注意一点：dp [i] [1]，**表示的是第i天，买入股票的状态，并不是说一定要第i天买入股票。**

达到dp [i] [1]状态，有两个具体操作：

- 操作一：第i天买入股票了，那么dp[i] [1] = dp[i-1] [0] - prices[i]
- 操作二：第i天没有操作，而是沿用前一天买入的状态，即：dp[i] [1] = dp[i - 1] [1]
- dp [i] [1]一定要选最大的
- dp[i] [1] = max(dp[i-1] [0] - prices[i], dp[i - 1] [1]);

同理dp[i] [2]也有两个操作：

- 操作一：第i天卖出股票了，那么dp[i] [2] = dp[i - 1] [1] + prices[i]
- 操作二：第i天没有操作，沿用前一天卖出股票的状态，即：dp[i] [2] = dp[i - 1] [2]
- dp[i] [2] = max(dp[i - 1] [1] + prices[i], dp[i - 1] [2])

同理：

- dp[i] [3] = max(dp[i - 1] [3], dp[i - 1] [2] - prices[i]);
- dp[i] [4] = max(dp[i - 1] [4], dp[i - 1] [3] + prices[i]);

3、dp数组如何初始化

第0天没有操作 dp[0] [0] = 0;

第0天做第一次买入的操作，dp[0] [1] = -prices[0];

第0天做第一次卖出的操作，这个初始值应该是多少呢?

首先卖出的操作一定是收获利润，整个股票买卖最差情况也就是没有盈利即全程无操作现金为0，

从递推公式中可以看出每次是取最大值，那么既然是收获利润如果比0还小了就没有必要收获这个利润了。

```
dp[0][2] = 0;
```

第二次买入依赖于第一次卖出的状态，其实相当于第0天第一次买入了，第一次卖出了，然后在买入一次（第二次买入），那么现在手头上没有现金，只要买入，现金就做相应的减少。

```
dp[0][3] = -prices[0];
dp[0][4] = 0;
```

4、确定遍历顺序

从递归公式其实已经可以看出，一定是从前向后遍历，因为dp[i]，依靠dp[i - 1]的数值

5、举例推导dp数组

以输入[1,2,3,4,5]为例

下标：	股票：	状态j：不操作	买入	卖出	买入	卖出
		0	1	2	3	4
0	1	0	-1	0	-1	0
1	2	0	-1	1	-1	1
2	3	0	-1	2	-1	2
3	4	0	-1	3	-1	3
4	5	0	-1	4	-1	4

题解

Python3

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # 动态规划
        if len(prices) == 0:
            return 0

        #初始化dp数组
        dp = [[0] * 5 for i in range(len(prices))]  #dp数组
        dp[0][1] = -prices[0]
        dp[0][3] = -prices[0]

        for i in range(1,len(prices)):
            dp[i][0] = dp [i-1][0]
            dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i])
            dp[i][2] = max(dp[i-1][2], dp[i-1][1] + prices[i])
            dp[i][3] = max(dp[i-1][3], dp[i-1][2] - prices[i])
            dp[i][4] = max(dp[i-1][4], dp[i-1][3] + prices[i])

        return dp[-1][4]
```

思考

ttttttttttttttttttt