121-买卖股票的最佳时机I

题述

121. 买卖股票的最佳时机

难度 简单 2266

给定一个数组 `prices` ，它的第 `i` 个元素 `prices[i]` 表示一支给定股票第 `i` 天的价格。

你只能选择 **某一天** 买入这只股票，并选择在 **未来的某一个不同的日子** 卖出该股票。设计一个算法来计算你所能获取的最大利润。

返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回 `0` 。

示例 1：

输入： [7,1,5,3,6,4]
输出： 5
解释： 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。

示例 2：

输入： prices = [7,6,4,3,1]
输出： 0
解释： 在这种情况下, 没有交易完成, 所以最大利润为 0。

思路

暴力

双循环寻找最大间距

动态规划

1、确定dp数组（dp table）以及下标的含义

dp[i] [0] 表示第i天持有股票所得最多现金

2、确定递推公式

如果第i天持有股票即dp[i] [0]， 那么可以由两个状态推出来

- 第i-1天就持有股票，那么就保持现状，所得现金就是昨天持有股票的所得现金 即：dp[i - 1] [0]
- 第i天买入股票，所得现金就是买入今天的股票后所得现金即：-prices[i]
- 那么dp[i] [0]应该选所得现金最大的，所以dp[i] [0] = max(dp[i - 1] [0], -prices[i]);

如果第i天不持有股票即dp[i] [1]， 也可以由两个状态推出来

- 第i-1天就不持有股票，那么就保持现状，所得现金就是昨天不持有股票的所得现金 即：dp[i - 1] [1]
- 第i天卖出股票，所得现金就是按照今天股票佳价格卖出后所得现金即：prices[i] + dp[i - 1] [0]
- 同样dp[i] [1]取最大的，dp[i] [1] = max(dp[i - 1] [1], prices[i] + dp[i - 1] [0]);

3、dp数组如何初始化

由递推公式 dp[i] [0] = max(dp[i - 1] [0], -prices[i]); 和 dp[i] [1] = max(dp[i - 1] [1], prices[i] + dp[i - 1] [0]);可以看出

其基础都是要从dp[0] [0]和dp[0] [1]推导出来。

那么dp[0] [0]表示第0天持有股票，此时的持有股票就一定是买入股票了，

因为不可能有前一天推出来，所以dp[0] [0] -= prices[0];

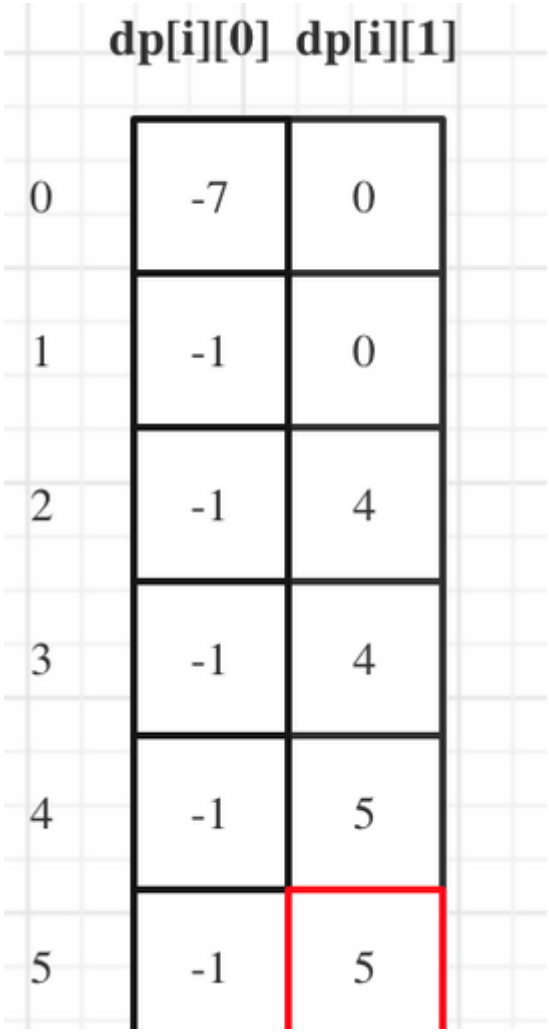
dp[0] [1]表示第0天不持有股票，不持有股票那么现金就是0，所以dp[0] [1] = 0;

4、确定遍历顺序

从递推公式可以看出dp[i]都是有dp[i - 1]推导出来的，那么一定是从前向后遍历。

5、举例推导dp数组

以示例1，输入：[7,1,5,3,6,4]为例，dp数组状态如下：



题解

Python

暴力

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # 暴力
        result = 0
        for i in range(len(prices)):
            j = i + 1
            for j in range(j,len(prices)):
                result = max(result,prices[j] - prices[i])
        return result
```

动态规划

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # 动态规划
        length = len(prices)
        if length == 0:
            return 0
        # 初始化dp数组
        dp = [[0] * 2 for i in range(length)]
```

```
        dp[0][0] = -prices[0]
        dp[0][1] = 0

        for i in range(1,length):
            dp[i][0] = max(dp[i-1][0] , -prices[i])
            dp[i][1] = max(dp[i-1][1] , prices[i] + dp[i-1][0])
        return dp[-1][1]
```

思考