0123. 买卖股票的最佳时机 III

ITCharge 大约 4 分钟

- 标签：数组、动态规划
- 难度：困难

题目链接

- 0123. 买卖股票的最佳时机 III - 力扣

题目大意

给定一个数组 `prices` 代表一只股票，其中 `prices[i]` 代表这只股票第 `i` 天的价格。最多可完成两笔交易，且不同同时参与躲避交易（必须在再次购买前出售掉之前的股票）。

现在要求：计算所能获取的最大利润。

解题思路

动态规划求解。

最多可完成两笔交易意味着总共有三种情况：买卖一次，买卖两次，不买卖。

具体到每一天结束总共有 5 种状态：

0. 未进行买卖状态；
1. 第一次买入状态；
2. 第一次卖出状态；
3. 第二次买入状态；
4. 第二次卖出状态。

所以我们可以定义状态 `dp[i][j]` ，表示为：第 `i` 天第 `j` 种情况（`0 <= j <= 4`）下，所获取的最大利润。

注意：这里第第 `j` 种情况，并不一定是这一天一定要买入或卖出，而是这一天所处于的买入卖出状态。比如说前一天是第一次买入，第二天没有操作，则第二天就沿用前一天的第一次买入状态。

接下来确定状态转移公式：

- 第 `0` 种状态下显然利润为 `0` ，可以直接赋值为昨天获取的最大利润，即 `dp[i][0] = dp[i - 1][0]` 。
- 第 `1` 种状态下可以有两种状态推出，取最大的那一种赋值：
  - 不做任何操作，直接沿用前一天买入状态所得的最大利润： `dp[i][1] = dp[i - 1][1]` 。
  - 第一次买入： `dp[i][1] = dp[i - 1][0] - prices[i]` 。
- 第 `2` 种状态下可以有两种状态推出，取最大的那一种赋值：
  - 不做任何操作，直接沿用前一天卖出状态所得的最大利润： `dp[i][2] = dp[i - 1][2]` 。
  - 第一次卖出： `dp[i][2] = dp[i - 1][1] + prices[i]` 。
- 第 `3` 种状态下可以有两种状态推出，取最大的那一种赋值：
  - 不做任何操作，直接沿用前一天买入状态所得的最大利润： `dp[i][3] = dp[i - 1][3]` 。
  - 第二次买入： `dp[i][3] = dp[i - 1][2] - prices[i]` 。
- 第 `4` 种状态下可以有两种状态推出，取最大的那一种赋值：
  - 不做任何操作，直接沿用前一天卖出状态所得的最大利润： `dp[i][4] = dp[i - 1][4]` 。
  - 第二次卖出： `dp[i][4] = dp[i - 1][3] + prices[i]` 。

下面确定初始化的边界值：

可以很明显看出第一天不做任何操作就是 `dp[0][0] = 0` ，第一次买入就是 `dp[0][1] = -prices[i]` 。

第一次卖出的话，可以视作为没有盈利（当天买卖，价格没有变化），即 `dp[0][2] = 0` 。第二次买入的话，就是 `dp[0][3] = -prices[i]` 。同理第二次卖出就是 `dp[0][4] = 0` 。

在递推结束后，最大利润肯定是无操作、第一次卖出、第二次卖出这三种情况里边，且为最大值。我们在维护的时候维护的是最大值，则第一次卖出、第二次卖出所获得的利润肯定大于等于 0。而且，如果最优情况为一笔交易，那么在转移状态时，我们允许在一天内进行两次交易，则一笔交易的状态可以转移至两笔交易。所以最终答案为 `dp[size - 1][4]` 。 `size` 为股票天数。

代码

```py
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        size = len(prices)
        if size == 0:
            return 0
        dp = [[0 for _ in range(5)] for _ in range(size)]

        dp[0][1] = -prices[0]
        dp[0][3] = -prices[0]

        for i in range(1, size):
            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[size - 1][4]
```