本日学习内容

1. 完成股票买卖问题全部，总结为博客发布
2. 完成学生管理系统的功能

本日分享内容

题目一：309. 买卖股票的最佳时机含冷冻期

- 给定一个整数数组`prices`，其中第 `prices[i]` 表示第 `*i*` 天的股票价格 。

  设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：

  - 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

  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 = [1,2,3,0,2]
  输出: 3
  解释: 对应的交易状态为: [买入, 卖出, 冷冻期, 买入, 卖出]
  ```

  示例 2:

  ```
  输入: prices = [1]
  输出: 0
  ```

  提示：

  - `1 <= prices.length <= 5000`
  - `0 <= prices[i] <= 1000`

思路

首先依旧是状态设计，我们给每一天设置三种状态：

dp[i][0]　//第i天不持股，且今天没有卖出，可以买入

dp[i][1]　//第i天持股

dp[i][2]　// 第i天不持股，但是今天刚刚卖出，正在冷冻期（明天不能买）
状态转移方程如下：

dp[i][0] = max(dp[i-1][0], dp[i-1][2]);　　// 休息：可以继续休息 或 从冷冻期恢复
dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i]); // 买入：只能从“可买入状态”买入
dp[i][2] = dp[i-1][1] + prices[i];　　// 卖出：必须从昨天持股中卖出
dp[0][0] = 0;　　// 第一天不操作
dp[0][1] = -prices[0];　// 第一天买入
dp[0][2] = 0;　　// 不可能在第一天就卖出（冷冻）

代码

```
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        if (n == 0) {
            return 0;
        }
        vector<vector<int>> dp(n, vector<int>(3, 0));

        dp[0][0] = 0;
        dp[0][1] = -prices[0];
        dp[0][2] = 0;

        for (int i = 1; i < n; ++i) {
            dp[i][0] = max(dp[i - 1][0], dp[i - 1][2]);         // 休息
            dp[i][1] = max(dp[i - 1][1], dp[i - 1][0] - prices[i]); // 买入
            dp[i][2] = dp[i - 1][1] + prices[i];                // 卖出
        }

        return max(dp[n - 1][0], dp[n - 1][2]);
    }
};
```

存储　　行 7，列 10

题目二：714. 买卖股票的最佳时机含手续费

给定一个整数数组 `prices`，其中 `prices[i]`表示第 `i` 天的股票价格 ； 整数 `fee` 代表了交易股票的手续费用。

你可以无限次地完成交易，但是你每笔交易都需要付手续费。如果你已经购买了一个股票，在卖出它之前你就不能再继续购买股票了。

返回获得利润的最大值。

注意：这里的一笔交易指买入持有并卖出股票的整个过程，每笔交易你只需要为支付一次手续费。

示例 1：

```
输入：prices = [1, 3, 2, 8, 4, 9], fee = 2
输出：8
解释：能够达到的最大利润：
在此处买入 prices[0] = 1
在此处卖出 prices[3] = 8
在此处买入 prices[4] = 4
在此处卖出 prices[5] = 9
总利润：((8 - 1) - 2) + ((9 - 4) - 2) = 8
```

示例 2：

```
输入：prices = [1,3,7,5,10,3], fee = 3
输出：6
```

提示：

- `1 <= prices.length <= 5 * 104`
- `1 <= prices[i] < 5 * 104`
- `0 <= fee < 5 * 104`

思路

我们可以继续使用和买卖股票问题2相似的思路，只不过每次交易要减去手续费。

状态定义：
dp[i][0]：第 i 天不持有股票的最大收益；

dp[i][1]：第 i 天持有股票的最大收益；

状态转移：
dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i] - fee)

解释：今天不持股，可以是昨天就不持股，或者昨天持股但今天卖了（收钱减手续费）；

dp[i][1] = max(dp[i-1][1], dp[i-1][0] - prices[i])

解释：今天持股，可以是昨天就持股，或者昨天不持股但今天买了（花钱买）；

初始值：
dp[0][0] = 0：第 0 天不持股，收益是 0；

dp[0][1] = -prices[0]：第 0 天持股，花了钱买入股票；

代码

```
public:
    int maxProfit(vector<int>& prices, int fee) {
        int n = prices.size();
        vector<vector<int>> dp(n, vector<int>(2, 0));

        // 初始化
        dp[0][0] = 0;               // 第 0 天不持股
        dp[0][1] = -prices[0];      // 第 0 天持股

        for (int i = 1; i < n; ++i) {
            // 今天不持股 = max(昨天不持股, 昨天持股 + 今天卖出的收益 - 手续费)
            dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + prices[i] - fee);

            // 今天持股 = max(昨天持股, 昨天不持股 - 今天买入花的钱)
            dp[i][1] = max(dp[i - 1][1], dp[i - 1][0] - prices[i]);
        }

        // 最后一天不能持股才能最大收益
        return dp[n - 1][0];
    }
};
```

存储 行 20，列 23

本日遇到的问题

1. 管理系统封装多文件遇到困难

明日学习内容

1. 继续测试管理系统的bug并修复
2. 开始复习随想录的前面的算法