算法导论

实
验
报
告

课　题 基于动态规划和贪心算法的股票最大利润求解方案

学　院 计算机学院

学　号 **1613415**

姓　名 潘巧巧

2019 年 **5** 月 **20** 日

目录

一、目的描述 1

二、实验内容 1

三、实验步骤 2

四、实验分析 6

五、参考资料 6

一、目的描述

股票购买是人们日常生活中非常重要的一部分。

本次实验聚焦于**已知股票走势时通过动态规划求解最大利润**的问题，将采用难度依次递增的方式，使用算法导论课上学习的**动态规划和贪心算法**相关知识对 6 个问题进行依次解答。

相关知识点：

1、 贪心算法

如果一个算法是通过一些小的步骤来建立一个解，在每一部根据局部情况选择一个决定使得某些主要的指标得到最优化，则该算法是贪心算法。对于一个问题常常可以设置许多不同的贪心算法，每个算法局部地、增量地优化在求解过程中的某些不同的量。

2、 动态规划

把目标分解成一系列子问题，然后对越来越大的子问题建立正确的解，从而隐含地探查所有可行解的空间。动态规划穿过问题可行解的指数规模的集合，但它甚至没有明确地检查所有解就做到了这一点。本实验主要采取**顺推法动态规划**实现。

二、实验内容

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。设计一个算法来计算分别满足以下各问题时所能获取的最大利润。

注意：同一时刻最多只能拥有一支股票，且必须先买入后卖出。

本实验按照难度递增依次解决以下 6 个问题

【问题 1-easy】全程最多**只允许完成一笔交易**（即买入和卖出该支股票）。

【问题 2-easy】全程可以为了最大利益完成**更多的交易**（多次买卖该支股票）且无手续费。

【问题 3-medium】全程可以为了最大利益完成**更多的交易**（多次买卖该支股票），但每次交易（一买一卖）需要缴纳一次**手续费**，注意同一时刻最多只能拥有一支股票。

【问题 4-medium】全程可以为了最大利益完成**更多的交易**（多次买卖该支股票），无手续费，但每次卖出股票后无法在第二天买入股票（即**冷冻期**为 1 天）。

【问题 5-hard】全程**最多完成两笔交易。**

【问题 6-hard】全程**最多完成 K 笔交易。**

三、实验步骤及实验结果

【问题 1】动态规划。

顺推公式：

最大利润 = max { 前一天最大利润，今天的价格 - 之前最低价格 }

```
int MaxProfit_1(const vector<int> prices) {
    if (prices.size() <= 1)
        return 0;
    int min_price = prices[0];
    int max_profit = 0;
    for (int i = 0; i < prices.size(); i++)
    {
        min_price = min(min_price, prices[i]);
        max_profit = max(max_profit, prices[i] - min_price);
    }
    return max_profit;
}
```

样例及说明：
输入：[7,1,5,3,6,4]
输出：5
解释：[无，买入，无，无，卖出，无]

【问题 2】贪心算法。

由于可以无限买入和卖出，且无手续费，故可以在同一天内卖出且买入，故只需要贪心地计算当天股票价格与前一天股票价格之差。

```
int MaxProfit_2(const vector<int> prices) {
    int max_profit = 0;
    for (int i = 1; i < prices.size(); i++) {
        if (prices[i] > prices[i - 1])
            max_profit += prices[i] - prices[i - 1];
    }
    return max_profit;
}
```

样例及说明：
输入：[7,1,5,3,6,4]
输出：7
解释：[无，买入，卖出，买入，卖出，无]
输入：[1,2,3,4,5]
输出：4
解释：[买入，卖出并买入，卖出并买入，卖出并买入，卖出]
//一天内可以重复操作（贪心）

【问题 3】贪心算法。

可以无限买入和卖出，但需要支付手续费，本应考虑尽量减少遍历过程中进入交易的次数，但通过巧妙设置 mini 的值为 prices[i]-fee 可以回避这一问题，进而使用贪心算法解决。

```
int MaxProfit_3(const vector<int> prices, int fee) {
    if (prices.size() < 2)
        return 0;
    int max_profit = 0;
    int mini = prices[0];
    for (int i = 0; i < prices.size(); i++)
    {
        if (prices[i] < mini) //当前的更小
            mini = prices[i];
        else if (prices[i] > mini + fee) //当前较大且有收益
        {
            max_profit += prices[i] - fee - mini;
            mini = prices[i] - fee;
        }
    }
    return max_profit;
}
```

输入：prices = [1, 3, 2, 8, 4, 9], fee = 2
输出：8
解释：[买入，卖出且买入，卖出且买入，卖出，买入，卖出]
总利润：((8 - 1) - 2) + ((9 - 4) - 2) = 8

【问题 4】动态规划。

顺推公式中的三个变量含义为：

sell [i]：截至第 i 天，最后一个操作是卖时的最大收益

buy [i]：截至第 i 天，最后一个操作是买时的最大收益

cool[i]：截至第 i 天，最后一个操作是冷冻期时的最大收益

```
int MaxProfit_4(const vector<int> prices)
{
    if (prices.size() < 2)
        return 0;
    vector<int> sell(prices.size());
    vector<int> cold(prices.size());
    vector<int> buy(prices.size());

    buy[0] = -prices[0];
    for (int i = 1; i < prices.size(); i++)
    {
        sell[i] = max(buy[i - 1] + prices[i], sell[i - 1]);
        buy[i]  = max(cold[i - 1] - prices[i], buy[i - 1]);
        cold[i] = max(max(sell[i - 1], buy[i - 1]), cold[i - 1]);
    }
    return sell[prices.size() - 1];
}
```

样例及说明：
输入：[1,2,3,0,2]
输出：3
解释：[买入，卖出，冷冻期，买入，卖出]

【问题 5】动态规划。

时间上有连续的定四个点，一买、一卖、二买、二卖。但允许当天同时按顺序进行这四手交易。每个定点找寻符合当前的最大值，由于状态是按顺序更新的，因此前值会影响后值。如果某个定点值没有更新，那么就意味着当前定点不变。

顺推式中，对于任意一天考虑四个变量:

FirstBuy: 在该天第一次买入股票可获得的最大收益

FirstSell: 在该天第一次卖出股票可获得的最大收益

SecondBuy: 在该天第二次买入股票可获得的最大收益

SecondSell: 在该天第二次卖出股票可获得的最大收益

分别对四个变量进行相应的更新，最后 SecondSell 就是最大收益值 max_profit

```
int MaxProfit_5(const vector<int> prices)
{
    int FirstBuy  = -prices[0], FirstSell = 0;
    int SecondBuy = -prices[0], SecondSell = 0;
    for(int i=0;i<prices.size();i++)
    {
        FirstBuy  = max(FirstBuy, -prices[i]);
        FirstSell  = max(FirstSell, FirstBuy + prices[i]);
        SecondBuy  = max(SecondBuy, FirstSell - prices[i]);
        SecondSell = max(SecondSell, SecondBuy + prices[i]);
    }
    int max_profit = SecondSell;
    return max_profit;
}
```

样例及说明
输入：[3,3,5,0,0,3,1,4]
输出：6
解释：[无，无，无，买入，无，卖出，买入，卖出]
输入：[1,2,3,4,5]
输出：4
解释：[买入，无，无，无，卖出]

【问题 6】动态规划+贪心算法

问题 6 为问题 5 的拓展，只需将问题 5 的 4 个变量拓展为二维数组（每维 2

个变量）即可解决问题。此外，注意到当 k 大于天数的一半时，直接采用问题 2 中的贪心算法可以大幅优化时间复杂度。

```
int MaxProfit_6(int k, vector<int> prices)
{
    if (prices.size() <= 1 || k == 0)
        return 0;
    if (k >= prices.size() / 2) // k大于等于数组长度一半时,问题转化为贪心问题
                                // 此时直接采用问题2的贪心方法解决可以大幅提升时间性能
        return MaxProfit_2(prices);
    vector<vector<int>> stock(k);
    for (int i = 0; i < k; i++)
    {
        stock[i].push_back(-prices[0]); // 在某天第i次买入股票可获得的最大收益
        stock[i].push_back(0);          // 在某天第i次卖出股票可获得的最大收益
    }
    for (int i = 0; i < prices.size(); i++)
    {
        stock[0][0] = max(stock[0][0], -prices[i]);
        stock[0][1] = max(stock[0][1], stock[0][0] + prices[i]);
        for (int j = 1; j < k; j++)
        {
            stock[j][0] = max(stock[j][0], stock[j - 1][1] - prices[i]);
            stock[j][1] = max(stock[j][1], stock[j][0] + prices[i]);
        }
    }
    int max_profit = stock[k - 1][1];
    return max_profit;
}
```

输入：[3,2,6,5,0,3]， k = 2
输出：7
解释：[无，买入，卖出，无，买入，卖出]

VS2019 下运行的实验结果：

四、实验分析

	时间复杂度	空间复杂度
问题一	O(n)	O(n)
问题二	O(n)	O(n)
问题三	O(n)	O(n)
问题四	O(n)	O(n)
问题五	O(n)	O(n)
问题六	O(mn)	O(n)

本实验利用贪心算法和顺推动态规划相结合，有效解决了已知股票走势时求最大利润的问题，且时间复杂度和空间复杂度均较为理想。可以有效地作为人们日常炒股和其他相关活动的参考。

五、参考资料

[1] 《算法设计》 Jon Kleinberg, Eva Tardos
[2] 《算法导论》 Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rives Clifford Stein