# 加餐 买卖股票：常见且必考的动态规划面试题

你好，我是卢誉声。

上一课我们介绍了动态规划面试问题中求方案总数和求可行性这两大类问题的通用解法，解题模版如下：

1. 根据特征判断是否用动态规划来解；
2. 确定初始化状态和状态参数；
3. 确定状态存储数组（即备忘录）；
4. 写出关键的状态转移方程；
5. 编写代码进行求解。

这样的解题模版（套路）是可以复用的，希望你能牢牢记住。今天，作为一节加餐课，我想给你介绍另一种常考的面试问题：买卖股票。这种问题的变种比较多，但依然可以用上述解题模版来解决所有买卖股票的问题，从而做到一通百通。

## 买卖股票问题

在技术面试环节，如果考察动态规划问题的话，那么买卖股票就是一类常考且经典的问题。这类问题一般来说属于求最优解（最大值和最小值）的范畴，下面我们看看这个问题到底是怎样的。

### 算法问题分析

问题：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。请你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：[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

示例2：

输入：[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

示例3：

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

可能对大部分人来说，第一眼看到这道题都没有什么思路——这个问题里存在什么可以提取的最优子结构吗？我来给你分析一下。

我们假设，一支股票某天在某种条件下（在前一天赚取的利润确定的情况下，这些条件肯定会影响我们的决策，因此暂时不考虑具体的条件到底是什么）已经赚取了利润。那么当天的利润是依赖于其前一天某些条件下的利润，所以这里存在重复计算问题，也就是会有**重叠子问题**。

而对于这支股票，前一天的利润会影响后一天的利润，反之是不会有影响的，那么这里**无后效性**的条件也满足了。

最后考虑一下，原问题要求解的是：最后一天结束时，一共赚取了多少利润。每天结束时，赚得的总利润 = 前一天赚取的总利润 ➕ 当天的决策（是否卖出或者买入股票），到这里我们终于嗅到了**最优子结构**的味道。

现在，我们有了一个大致的思路：这是一个可以使用动态规划求解的问题。现在，再来看一下这个问题的状态转移方程到底是什么？

## 写出状态转移方程

首先，我们要确定**初始化状态**，思考一下有哪些边界情况。

1. 第一种边界情况是：如果当天结束时没有持股，而且到当天结束时从未卖出过股票，这种情况利润肯定为 0；
2. 第二种边界情况是：当天持股，而且卖出过两次股票，这种情况是不存在的，这也就是我们的终止条件。

然后，可以看出这个问题的**状态参数**之一是天数（变量），也就是当前是第几天，毕竟没有天数也就没有我们的子问题。

根据前面的分析，得知问题的形式是前一天赚取的总利润 ➕ 当天的决策——也就是在当天结束时是否持有这支股票，以及我们当天买卖当前这支股票（每支股票最多只能买卖两次），进而确定当天结束时赚取的总利润。

现在我们得知，在每一天结束时的总利润取决于以下三个因素：

1. 前一天赚取的总利润；
2. 当天结束时是否持有股票；
3. 当天是否买进或者卖出股票。

此时，由于买卖股票是有次数限制的，即只能有2次。因此，对于第三个因素，我们需要进一步具体化才能做出决策——未卖出过股票、卖出过一次股票和卖出过两次股票。

所以，这里我们就找出了三个状态参数，它们决定了某一天结束时得到的总利润，分别是：

1. 天数；
2. 当天结束时是否持有股票；
3. 股票卖出的次数。

根据这三个状态参数（因素），再结合前一天赚取的总利润，就可以得到当前这一天这个状态下的最优解了。只不过前一天赚取的总利润肯定也会受到这三个参数的影响。为此，还需要根据当天的参数来思考前一天的参数有哪些可能性，才能知道使用前一天哪种状态下的总利润（毕竟有三个参数）。

经过上面对状态参数的分析，我们可以知道状态存储空间，即备忘录是一个三维数组 **\(DP\[i\]\[j\]\[k\]\)，表示在第 \(i\) 天，是否持有（其中 \(j\) 为 0 表示未持有，1 表示持有）以及卖出了几次（其中 \(k\) 为 0 表示卖出 0 次，1 表示卖出 1 次，2 表示卖出 2 次）股票的情况下，最大利润是多少。**

基于以上分析，我们就可以写出**状态转移方程**了。

\[DP(i, j, k)=\\left\\{\\begin{array}{c}- 0, & case1: j=0\\ and\\ k=0 \\\\\\- max(DP\[i-1\]\[1\]\[0\]+p\[i\], DP\[i-1\]\[0\]\[1\]), & case2: j=0\\ and\\ k=1 \\\\\\- max(DP\[i-1\]\[1\]\[1\]+p\[i\], DP\[i-1\]\[0\]\[2\]), & case3: j=0\\ and\\ k=2 \\\\\\- max(DP\[i-1\]\[0\]\[0\]-p\[i\], DP\[i-1\]\[1\]\[0\]), & case4: j=1\\ and\\ k=0 \\\\\\- max(DP\[i-1\]\[0\]\[1\]-p\[i\], DP\[i-1\]\[1\]\[1\]), & case5: j=1\\ and\\ k=1 \\\\\\- \-INF, & case5: j=1\\ and\\ k=2 \\-\\end{array}\\right.\]

这个状态转移方程比较复杂，需要进一步解释。

1. 初始化状态，如果当天未持股，而且到当天结束时从未卖出过股票，利润必定为 0。
2. 如果当日结束时未持股，卖出过 1 次股票。那么在这种情况下，可能是今天卖出；也可能是之前卖出的，所以当天的利润可能存在两种情况。同时，我们需要从这两种情况中取最大值作为当天的最大利润：

> a. 股票是当天卖出的；- b. 股票是前一天卖出的（前一天肯定是未持股状态，而且已经卖出过 1 次股票）。

1. 如果当日结束时未持股，卖出过 2 次股票。那么在这种情况下，可能是今天卖出；也可能是之前卖出的，所以当天的利润可能存在两种情况。同时，我们需要从这两种情况中取最大值作为当天的最大利润：

> a. 股票是当天卖的；- b. 股票是前一天已经卖出的（前一天肯定是未持股状态，而且已经卖出过 2 次股票）。

1. 如果当日结束时持股，未卖出过股票。那么在这种情况下，可能是今天买进；也可能是之前买进的，所以当天的利润可能存在两种情况：

> a. 股票是当天买进的；- b. 股票是前一天已经买进的（前一天肯定是持股状态，而且从未卖出过股票）；- 因此，当天的最大利润就是从这两种情况中取最大值。需要注意的是，由于这里是买进股票的情况，所以如果当日买进了股票，那么利润需要减去当天的股票价值。

1. 如果当日结束时持股，卖出过 1 次股票。那么在这种情况下，可能是今天买进；也可能是之前买进的，所以当天的利润可能存在两种情况：

> a. 股票是当天买进的；- b. 股票是前一天已经买进的（前一天肯定是持股状态，而且卖出过 1 次股票）；- 因此，当天的最大利润就是从这两种情况中取最大值。需要注意的是，由于这里是买进股票的情况，所以如果当日买进了股票，那么利润需要减去当天的股票价值。

1. 如果当日结束后持有股票，卖出过 2 次股票。这种情况不存在，直接设置为 -INF （代码中要做对应的处理）。

分析完毕，需要注意的情况比较多。你也会发现考察这类问题时，需要细心，不要遗漏掉原问题给出的条件，同时要注意卖出和买进之间的关系。

## 编写代码进行求解

写出该问题的状态转移方程，我们的工程就完成一大半了。现在，我给出求解该问题的代码实现。

Java实现：

```
int getMaxStock(int[] prices) {
    int m = prices.length;
    int dp[][][] = new int[m][2][3];

    // 处理第一天
    // 假设第一天没有买入
    dp[0][0][0] = 0;
    dp[0][0][1] = 0;
    dp[0][0][2] = 0;

    // 第一天不可能已卖出
    dp[0][1][0] = -prices[0];
    dp[0][1][1] = -prices[0];
    dp[0][1][2] = -prices[0];

    // 处理后续日期
    for (int i = 1; i < m; i ++) {
        dp[i][0][0] = 0;
        dp[i][0][1] = Math.max(dp[i - 1][1][0] + prices[i], dp[i - 1][0][1]);
        dp[i][0][2] = Math.max(dp[i - 1][1][1] + prices[i], dp[i - 1][0][2]);
        dp[i][1][0] = Math.max(dp[i - 1][0][0] - prices[i], dp[i - 1][1][0]);
        dp[i][1][1] = Math.max(dp[i - 1][0][1] - prices[i], dp[i - 1][1][1]);
        dp[i][1][2] = 0;
    }

    return Math.max(dp[m - 1][0][1], dp[m - 1][0][2]); // 输出答案
}
```

C++实现：

```
int GetMaxStock(const vector<int>& prices) {
    int m = prices.size();
    int dp[m][2][3];

    // 处理第一天
    // 假设第一天没有买入
    dp[0][0][0] = 0;
    dp[0][0][1] = 0;
    dp[0][0][2] = 0;

    // 第一天不可能已卖出
    dp[0][1][0] = -prices[0];
    dp[0][1][1] = -prices[0];
    dp[0][1][2] = -prices[0];

    // 处理后续日期
```

```
    for (int i = 1; i < m; i ++) {
        dp[i][0][0] = 0;
        dp[i][0][1] = max(dp[i - 1][1][0] + prices[i], dp[i - 1][0][1]);
        dp[i][0][2] = max(dp[i - 1][1][1] + prices[i], dp[i - 1][0][2]);
        dp[i][1][0] = max(dp[i - 1][0][0] - prices[i], dp[i - 1][1][0]);
        dp[i][1][1] = max(dp[i - 1][0][1] - prices[i], dp[i - 1][1][1]);
        dp[i][1][2] = 0;
    }

    return max(dp[m - 1][0][1], dp[m - 1][0][2]); // 输出答案
}
```

通过比较状态转移方程和代码实现，我们发现实现股票买卖问题的代码还是比较容易的。基本上，就是照搬状态转移方程中的状态转移定义。

## 攻破买卖股票问题的解题模板

在讨论了具体的买卖股票问题之后，你就会发现，买卖股票问题的条件设定比较灵活多变（比如问题中可能限定只能买卖一次，卖出一次之后可能需要等待一定时间才能买入），也就是有交易冷冻期，每次交易需要支付手续费等。稍作修改就可以变成另一道题。

因此，我们说买卖股票问题是一类容易考察的问题，我们很有必要提炼出攻破该类问题的解题模板（套路）。

### 经过经验总结的解题模板

我们可以这样描述买卖股票类型的问题。

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 \(k\) 笔交易**。附加条件是：

1. 每次卖出股票**之后 \(t\) 天内**你无法进行任何交易，同时买入股票的时候会**收取 \(c\) 元的交易手续费**；
2. 你不能同时参与多笔交易，即你必须在再次购买前出售掉之前的股票。

### 对解题模板进行分析

相比于前面我讲的具体的买卖股票问题，这个解题模板里多了这么几个要素：

1. 最多 2 笔交易变成了 \(k\) 笔交易；
2. 多了一个交易冻结期限制，即 \(t\) 天之内无法进行任何交易；
3. 买入股票可能需要交易手续费，即卖出股票的时候需要支付额外的费用。

这几个因素产生的影响有：

1. 原本需要计算的是 2 次交易的最优解，现在需要求 $k$ 次交易的最优解；
2. 原本只需要在前一天的基础上进行决策，现在由于存在冻结期 $t$。因此，卖出或买进股票时需要在冻结期之前进行决策，而不是前一天；
3. 由于多了手续费 $c$，因此买入股票的时候需要扣掉手续费。

从表面上看，解题模板比上面的问题更复杂。但如果仔细思考一下，其实整个问题的框架并没有什么实质性变化。

待解的问题依然是：确定每天结束时的最大利润。但是，由于原问题里多了一个交易冻结期 $t$ 的限制。因此，我们需要考虑的问题就变多了：不仅要在前一天的基础上做出决策，还需要考虑冻结期的时间。

至于 $k$ 笔交易和手续费 $c$，则不影响整个问题的解题框架。

在解题模板中，由于待解问题的核心不变，所以重叠子问题、无后效性和最优子结构，则与之前的问题没有变化，因此不再赘述。

## 写出解题模板的状态转移方程

对于解题模板中多出来的这些因素，都不会影响状态参数。因此状态参数没有发生变化，分别是：

1. 天数；
2. 当天结束时是否持有股票；
3. 股票卖出的次数。

接着，我们来考虑状态存储，即备忘录的设计问题。由于现在交易次数上限从 2 次变成了 $k$ 次，因此状态存储空间需要改变。

在前面的具体买卖股票问题中，交易次数的上限是 2 次。那时，状态存储空间是三维数组 $DP[i][2][3]$，其中第三个维度表示股票卖出次数。那么，如果交易上限变成 $k$ 次，状态转移数组就变成了 **$DP[i][j][k+1]$，表示在第 $i$ 天，是否持有（其中 $j$ 为 0 表示未持有，1 表示持有）以及卖出了几次（其中 $k$ 为 0 表示卖出 0 次，1 表示卖出 1 次，2 表示卖出 2 次 ... 以此类推）股票的情况下，最大利润是多少。**

此外，我们还要考虑一下，求解这个问题存在哪些边界情况：

1. 第一种边界情况没有变化：如果当天结束时没有持股而且到当天结束时从未卖出过股票，这种情况利润肯定为0；
2. 第二种边界情况发生了变化：由于交易次数限制从 2 次变成了 \(k\) 次，因此这里边界变成：当天持股，而且卖出过 \(k\) 次股票，而对于情况不存在的，利润设定为负无穷（实际情况下可能需要在编写代码时进行调整）。

我们发现这个问题的状态参数基本没有发生改变，只有交易上限 \(k\) 影响了状态存储和初始化参数。现在，给出状态转移方程。

\[DP(i, j, k)=\\left\\{\\begin{array}{c}- 0, & case1: j=0\\ and\\ k=0 \\\\\\- max(DP\[i-1\]\[1\]\[k-1\]+p\[i\], DP\[i-1\]\[0\]\[k\]), & case2: j=0\\ and\\ k<=k_{max} \\\\\\- max(DP\[i-1-t\]\[0\]\[k\]-p\[i\]-c, DP\[i-1\]\[1\]\[k\]), & case4: j=1\\ and\\ k<k_{max} \\\\\\- \-INF, & case5: j=1\\ and\\ k=k_{max} \\\\\\- \\end{array}\\right.\]

同理，这个状态转移方程比较复杂。因此，我这里对其作出解释。

1. 初始化状态，如果当天未持股，而且到当天结束时从未卖出过股票，利润必定为0。
2. 如果当日结束时未持股，卖出过 \(k\) 次股票。那么在这种情况下，可能是今天卖出；也可能是之前卖出的，所以当天的利润可能存在两种情况。同时，我们需要从这两种情况中取最大值作为当天的最大利润：

> a. 股票是当前卖出的（前一天肯定是持股状态，而且已经卖出过 \(k-1\) 次股票）；- b. 股票是前一天已经卖出的（前一天肯定是未持股状态，而且已经卖出过 \(k\) 次股票）。

1. 如果当日结束时持股，卖出过 \(k\) 次股票。那么在这种情况下，可能是今天买进；也可能是之前买进的，所以当天的利润可能存在两种情况。同时，我们需要从这两种情况中取最大值作为当天的最大利润：

> a. 股票是当天买进的（前 \(t+1\) 天肯定是持股状态，而且已经卖出过 \(k-1\) 次股票），这里需要考虑 \(t\) 天的冻结期，\(t\) 天之内无法交易的，所以上一个状态是 \((1+t)\) 天之前，而不是 1 天前；- b. 股票是前一天已经买进的（前一天肯定是持股状态，而且卖出过1次股票）；- 因此，当天的最大利润就是从这两种情况中取最大值。需要注意的是，由于这里是买进股票的情况。所以，如果当日买进了股票，那么利润需要减去当天的股票价值。另外，由于我们可能涉及 \(c\) 元的手续费，因此这里买入的时候需要扣去 \(c\) 元的手续费，相当于股票的购入价格上升。

1. 如果当日结束后持有股票，卖出过 \(K_{max}\) 次股票，这种情况不存在，直接设置为 INF（编码时需要考虑这个怎么处理）。

这样我们就能求出最后一天的最优解了。其实，冻结期 \(t\) 和 \(c\) 元手续费只影响了问题中的部分参数，比如冻结期 \(t\) 影响了在买入股票时的状态转移参数（从 -1 变成了\(-(1+t)\)）；而手续费 \(c\) 则影响了买入股票时的成本（多减去了 \(c\) 元）；而最大售出次数则影响了边界条件。

### 实例化解题模板

现在，我们看一个实例化解题模板后的具体问题。

问题是这样的：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。请你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 3 笔交易**。附加条件是：

1. 每次买入股票的时候会**收取 2 元的交易手续费**；
2. 你不能同时参与多笔交易，即你必须在再次购买前出售掉之前的股票。

根据前面的分析得知，**状态参数**有三个：天数、当天结束时是否持有股票、股票卖出的次数。对状态存储，即备忘录来说 **\(DP\[i\]\[2\]\[4\]\) 表示在第 \(i\) 天，是否持有以及卖出了几次股票（最多 3 笔交易）的情况下，最大利润是多少。**

在写出状态转移方程前，再考虑一下初始化状态：

1. 第一种边界情况：如果当天结束时没有持股而且到当天结束时从未卖出过股票，这种情况利润肯定为 0；
2. 第二种边界情况：当天持股，而且卖出过3次股票，这种情况不存在的，利润设定为 -INF（实际情况下可能需要编码时调整）。

最后，我们根据以上信息给出了状态转移方程：

\[DP(i, j, k)=\\left\\{\\begin{array}{c}- 0, & case1: j=0\\ and\\ k=0 \\\\\\- max(DP(i-1,1,k-1)+p\[i\], DP(i-1,0,k)), & case2: j=0\\ and\\ k<=3 \\\\\\- max(DP(i-1,0,k)-p\[i\]-2, DP(i-1,1,k)), & case4: j=1\\ and\\ k<3 \\\\\\- \-INF, & case5: j=1\\ and\\ k=3 \\\\\\- \\end{array}\\right.\]

这里，我们把最大次数 \(K_{max}\) 替换成了 3，把冻结期 \(t\) 替换成 0，把手续费 \(c\) 替换成 2。通过买卖股票的解题模板，我们就能非常轻松地解决这些问题了。

## 课程总结

鉴于我们刚刚已经总结了解题模版，这里就不再赘述了。最后再啰嗦一句吧，其实很多动态规划问题就像我们处理股票问题的框架一样，很多类似的题目都可以通过总结分析，直接套用模

板，效果会非常好！你不妨多去试试。

## 课后思考

你能否写出通用的买卖股票的代码实现。另外，请你思考一下是否存在时间或空间复杂度优化的可能性？

期待你的分享，任何问题欢迎来留言区一起讨论！

上一页 下一页