只能买卖一次的股票最佳时机

121. 买卖股票的最佳时机

题目链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/

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。

贪心算法：

> 很自然就是取**最左最小值**，取**最右最大值,那么得到的差值就是最大利润。**

```
class Solution {
public:
 //low表示下标0...i的prices中的最低价格,因此由局部最优求得全局最优
    int maxProfit(vector<int>& prices) {
 int low=INT_MAX;
 int maxResult=0;
 for(int i=0;i<prices.size();i++){
 low=min(low,prices[i]);
 maxResult=max(maxResult,prices[i]-low);
  }
  return maxResult;
     }
};
```

动态规划方法

> 本道题的动态规划方法中

1、dp数组及其下标含义
dp[i]表示从下标0....i的价格中买卖股票获取的最大利润
2、递推公式
dp[i]=max(dp[i-1],prices[i]-low);
3、初始化
dp[0]=0;low=prices[0];
dp[1]=max(dp[0],prices[1]-low);
4遍历顺序:从前向后遍历

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
 if(prices.size()==0) return 0;
 int low=prices[0];
 vector<int> dp(prices.size(),0);
 for(int i=1;i<prices.size();i++){
 low=min(low,prices[i]);
 dp[i]=max(dp[i-1],prices[i]-low);
  }
  return dp[prices.size()-1];
     }
};
```

可以无数次买卖股票的最佳时机

122.买卖股票的最佳时机II

题目链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/

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你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

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）。

1这里重申一下dp数组的含义：
dp[i][0]表示第i天持有股票所得现金
dp[i][1]表示第i天不持有股票所得最多现金
2递推公式：
第i天持有股票所得最大现金
dp[i][0]=max(dp[i-1][0],dp[i-1][1]-prices[i]);
第i天不持有股票所获得最大现金

> **dp[i][1]=max(dp[i-1][1],dp[i-1][0]+prices[i]);**
> 3遍历顺序,根据天数从小天到大天
> 4初始化
> dp[0][0]=-prices[0];
> dp[0][1]=0;

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
 //注意dp数组的含义第二维度0表示持有，1表示不持有
 vector<vector<int>> dp(prices.size(),vector<int>(2,0));
 dp[0][0]=-prices[0];
 dp[0][1]=0;
 int len=prices.size();
 for(int i=1;i<len;i++)P{
  dp[i][0]=max(dp[i-1][0],dp[i-1][1]-prices[i]);
  dp[i][1]=max(dp[i-1][1],dp[i-1][0]+prices[i]);
  }
  return max(dp[len-1][0],dp[len-1][1]);//实际上只要返回dp[len-1][1]不持有的情况即可
    }
};
```

滚动数组降低空间复杂度

> 对于动态规划问题可以用滚动数组降低空间复杂度
> **滚动数组的形式有两种其一:直接有二维变为一维，其二：仍然是二维但是只记录当前数据和之前一个数据**

```
int maxProfit(vector<int>& prices){
 //定义滚动数组则是将第一个维度直接降为2
 vector<vector<int>> dp(2,vector<int>(2,0));
 dp[0][0]-=prices[0];//持有
 dp[0][1]=0;//不持有
 int len=prices.size();
 for(int i=1;i<len;i++){
 dp[i%2][0]=max(dp[(i-1)%2][0],dp[(i-1)%2][1]-prices[i]);
 dp[i%2][1]=max(dp[(i-1)%2][1],dp[(i-1)%2][1]+prices[i]);
  }
  return dp[(len-1)%2][1];
}
```

限制股票买卖时间仅限两次

123.买卖股票的最佳时机III

题目链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iii/

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 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：prices = [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，这笔交易所能获得利润 = 3-0 = 3 。随后，在第 7 天（股票价格 = 1）的时候买入，在第 8 天 （股票价格 = 4）的时候卖出，这笔交易所能获得利润 = 4-1 = 3。

> 关键在于至多买卖两次,这意味着可以买卖一次，可以买卖两次，也可以买卖0次
> 1、确定dp数组以及下标的含义
> 0、表示从始至终没有操作，1、第一次买入 2、第一次卖出
> 3、第二次买入4、第二次卖出(**注意:其中买入和卖出分别表示持有股票以及不持有股票的意思**)
> dp[i][j]中i表示第i天,j为[0-4]五个状态,dp[i][j]表示第i天状态j所剩下最大现金
> 2、确定递推公式
> 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]);
> 最终的结果是return max(dp[len-1][2],dp[len-1][4]);
> 现在最大的时候一定是卖出的状态，而两次卖出的状态现金最大的一定是最后一次卖出。
> 3初始化结果
> dp[0][0]=0;//什么都不操作当然是0
> dp[0][1]-=prices[0];//买入操作就是减少现金
> dp[0][2]=0;
> dp[0][3]-=prices[0];//不用管第几次，现在手头上没有现金，只要买入，现金就做相应的减少。
> dp[0][4]=0;
> 4、从递推公式可以知道是从前向后遍历,**因为dp[i]依靠dp[i-1];**

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
 if(prices.size()==0) return 0;
 vector<vector<int>>dp(prices.size(),vector<int>(5,0));
 dp[0][0]=0;dp[0][1]-=prices[0];
```

```
dp[0][2]=0;dp[0][3]-=prices[0];dp[0][4]=0;
for(int i=1;i<prices.size();i++){
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 max(dp[prices.size()-1][2],dp[prices.size()-1][4]);
    }
};
```

滑动数组解法：

```
int maxProfit(vector<int>& prices) {
 if(prices.size()==0) return 0;
 vector<int>dp(5,0);
 dp[0]=0;dp[1]-=prices[0];
 dp[2]=0;dp[3]-=prices[0];dp[4]=0;
 for(int i=1;i<prices.size();i++){
 dp[4]=max(dp[4],dp[3]+prices[i]);
 dp[3]=max(dp[3],dp[2]-prices[i]);
 dp[2]=max(dp[2],dp[1]+prices[i]);
 dp[1]=max(dp[1],dp[0]-prices[i]);
 }
 return max(dp[2],dp[4]);
    }
```

188.买卖股票的最佳时机IV

题目链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iv/

给定一个整数数组 prices ，它的第 i 个元素 prices[i] 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。

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：k = 2, prices = [2,4,1]
输出：2
解释：在第 1 天 (股票价格 = 2) 的时候买入，在第 2 天 (股票价格 = 4) 的时候卖出，这笔交易所能获得利润 = 4-2 = 2。

> 对于该问题相对于上面问题就是买卖次数不是一个定值而是一个变量
> 根据上面那道题的归纳分析法可以知道:如果j为奇数表示持有,j为偶数表示不持有
> dp[i][j]=max(dp[i-1][j],dp[i-1][j-1]+pow(-1,j%2)*prices[j]);
> 初始化
> dp[0][j]=j%2?-1*prices[0]:0;

```
1
2  class Solution {
3  public:
4      int maxProfit(int k, vector<int>& prices) {
5     if(prices.size()==0) return 0;
6          vector<vector<int>> dp(prices.size(),vector<int>(2*k+1,0));
7          //初始化
8          for(int i=0;i<=2*k;i++) dp[0][i]=i%2?-1*prices[0]:0;
9          //开始遍历
10          for(int i=1;i<prices.size();i++){
11              for(int j=1;j<=2*k;j++){
12                  dp[i][j]=dp[i-1][j]>=(dp[i-1][j-
1]+pow(-1,j%2)*prices[i])?dp[i-1][j]:dp[i-1][j-1]+pow(-1,j%2)*prices[i];
13              }
14          }
```

```
        return dp[prices.size()-1][2*k];
    }
};
```

309.最佳买卖股票时机含冷冻期

题目链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/

给定一个整数数组，其中第 i 个元素代表了第 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）：

- 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）。
- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

这道题相对于之前的题目就是买卖股票有条件
dp[i][0]表示第i天是不持有股票最大现金
dp[i][1]表示第i天持有股票的最大现金

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
 vector<vector<int>> dp(prices.size(),vector<int>(2,0));
 if(prices.size()==0) return 0;
 if(prices.size()==1) return 0;
 if(prices.size()==2) return max(0,prices[1]-prices[0]);
 dp[0][1]-=prices[0];
 dp[1][0]=max(0,prices[1]-prices[0]);
  dp[1][1]=max(dp[0][1],-1*prices[1]);
  for(int i=2;i<prices.size();i++){
  dp[i][0]=max(dp[i-1][0],dp[i-1][1]+prices[i]);
  dp[i][1]=max(dp[i-1][1],dp[i-2][0]-prices[i]);
  }
  return dp[prices.size()-1][0];
    }
};
```

714.买卖股票的最佳时机含手续费

题目链接：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/

给定一个整数数组 prices，其中第 i 个元素代表了第 i 天的股票价格 ；非负整数 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。