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）。

示例 1:

```
输入: [7,1,5,3,6,4]
输出: 7
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
     随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6-3 = 3 。
```

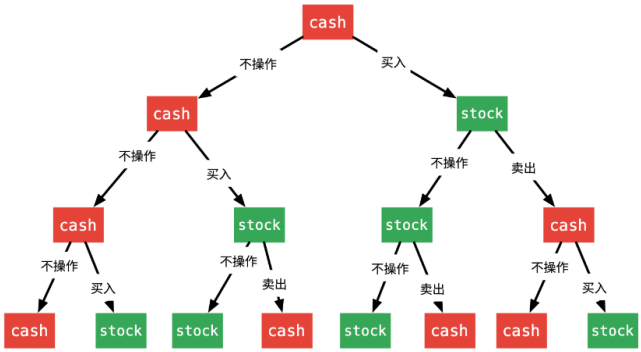
示例 2:

```
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，这笔交易所能获得利润 = 5-1 = 4 。
     注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。
     因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。
```

示例 3:

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

1 直接搜索暴力求解



2 贪心算法

贪心算法，**每走一步都做出当时看起来最佳的选择**，也就是说，总是做出局部最优解，希望这样的选择能导致全局最优解

注：

1 最佳可能是最大也可能是最小，看具体情况

2 只考虑接下来的一步，不考虑前和后，时间复杂度一般是线性，空间复杂的一般是常数

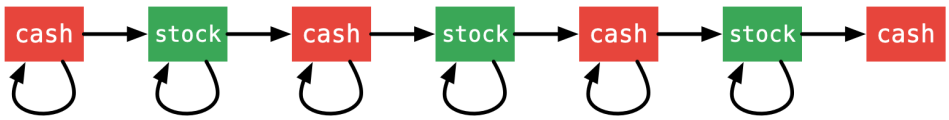
思路：
从第 i 天（这里 i >= 1）开始，与第 i - 1 的股价进行比较，如果股价有上升（严格上升），就将升高的股价（ prices[i] - prices[i- 1] ）记入总利润，按照这种算法，得到的结果就是符合题意的最大利润。
今天股价 - 昨天股价 有3种可能：正，负，0，那么只加最好的 **正数**
为什么这道题贪心算法是最优解：
加 0， 那么利润不增不减
加正数，利润会增加
加负数，利润只能减少

```
def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        res  =  0
        size = len(prices)
        for i in range(1,size):
            res += max(prices[i] - prices[i-1], 0)
         return res
```

时间复杂度：O(N)
空间复杂度：O(1)
3 动态规划：设置一个二维矩阵
1 定义状态：
dp[i][j]：
i：索引为 i的那一天能获得最大利润，
j: 索引为i的那个一天是股票还是现金， 现金为 0， 股价为1

2 状态转移方程：

- 状态从持有现金（cash）开始，到最后一天我们关心的状态依然是持有现金（cash）；
- 每一天状态可以转移，也可以不动。状态转移用下图表示：



说明：
因为不限制交易次数，除了最后一天，每一天的状态可能不变化，也可能转移；
写代码的时候，可以不用对最后一天单独处理，输出最后一天，状态为 0 的时候的值即可。
3 确定起始：
开始：
什么都不做：dp[0][0]
如果买入：dp[0][1] = -prices[i]
4 确定终止：

dp[len-1][0]

```
def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        size = len(prices)
        if len < 2:
            return 0
        # 0: 持有现金
         # 1: 持有股票
         # 状态转移:  0->1->0->1->0...
         dp = [[0,0]] * size
         dp[0][0] = 0
         dp[0][1] = -prices[0]
         for i in range(1,size):
             #是选择不变，还是卖出
             dp[i][0] = max(dp[i-1][0],dp[i-1][1] + prices[i])
             # 是选择不变，还是买入
             dp[i][1] = max(dp[i-1][1],dp[i-1][0] - prices[i])
         return dp[size-1][0]
```

时间复杂度 O(N)

空间复杂度 O(1)

```
def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        size = len(prices)
        if len < 2:
            return 0
        # 0: 持有现金
         # 1: 持有股票
         # 状态转移:  0->1->0->1->0...

         cash = 0
         hold = -prices[0]

         precash = cash
         prehold = hold
         for i in range(1,size):
```

```
            cash = max(precash, prehold + prices[i])
            hold = max(prehold, precash - prices[i])
            precash = cash
            prehold = hold
        return cash
```