122 Best Time to Buy and Sell Stock II

2018年4月1日 17:26

Question:

Say you have an array for which the i^{th} element is the price of a given stock on day *i*.
Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

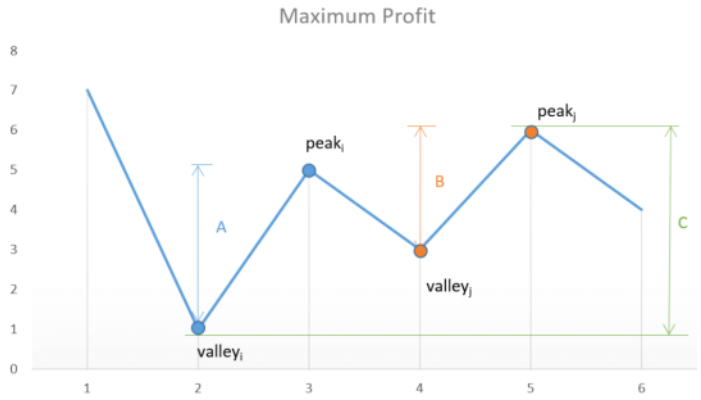
来自 <https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/description/>

假设有一个数组，它的第 i 个元素是一个给定的股票在第 i 天的价格。
设计一个算法来找到最大的利润。你可以完成尽可能多的交易（多次买卖股票）。然而，你不能同时参与多个交易（你必须在再次购买前出售股票）。

Say the given array is:

[7, 1, 5, 3, 6, 4].

If we plot the numbers of the given array on a graph, we get:



If we analyze the graph, we notice that the points of interest are the consecutive valleys and peaks.

Mathematically speaking:

$$TotalProfit = \sum_i (height(peak_i) - height(valley_i))$$

[1, 7, 2, 3, 6, 7, 6, 7]

The graph corresponding to this array is:

From the above graph, we can observe that the sum $A + B + C$ is equal to the difference D corresponding to the difference between the heights of the consecutive peak and valley.

Solution for Python3:

```python
# 最近的山峰-山谷:时间复杂度O(n), 空间复杂度O(1)
class Solution1:
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if not prices:
            return 0
        i, maxprofit, valley, peak = 0, 0, prices[0], prices[0]
        while i < len(prices) - 1:
            while i < len(prices) - 1 and prices[i] >= prices[i + 1]:
                i += 1
            valley = prices[i]
            while i < len(prices) - 1 and prices[i] <= prices[i + 1]:
                i += 1
            peak = prices[i]
            maxprofit += peak - valley
        return maxprofit


class Solution2:
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        maxprofit = 0
        for i in range(1, len(prices)):
            if prices[i] > prices[i - 1]:
                maxprofit += prices[i] - prices[i - 1]
        return maxprofit
```

Solution for C++:

```cpp
class Solution1 {
public:
    int maxProfit(vector<int>& prices) {
        if (prices.empty()) {
           return 0;
        }
        int i = 0, maxprofit = 0, valley = prices[0], peak = prices[0];
        while (i < prices.size() - 1) {
           while ((i < prices.size() - 1) && prices[i] >= prices[i + 1]) {
               i++;
           }
           valley = prices[i];
           while ((i < prices.size() - 1) && prices[i] <= prices[i + 1]) {
               i++;
           }
           peak = prices[i];
           maxprofit += peak - valley;
        }
```

```
        return maxprofit;
    }
};

class Solution2 {
public:
    int maxProfit(vector<int>& prices) {
        int maxprofit = 0;
        for (int i = 1; i < prices.size(); i++) {
           if (prices[i] > prices[i - 1]) {
               maxprofit += prices[i] - prices[i - 1];
           }
        }
        return maxprofit;
    }
};
```