123. Best Time to Buy and Sell Stock III

Hard 6243 124 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i^th` day.

Find the maximum profit you can achieve. You may complete **at most two transactions**.

Note: You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: prices = [3,3,5,0,0,3,1,4]
Output: 6
Explanation: Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.
```

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int hold[n][3];
        int not_hold[n][3];

        memset(hold, -1000000, sizeof(hold));
        memset(not_hold, -1000000, sizeof(hold));

        hold[0][0] = -prices[0];
        not_hold[0][0] = 0;

        for (int i = 1; i < n; i++) {
            for (int j = 0; j < 3; j ++) {
                hold[i][j] = max(hold[i - 1][j], not_hold[i - 1][j] - prices[i]);
                not_hold[i][j] = not_hold[i - 1][j];
                if (j - 1 >= 0) not_hold[i][j] = max(not_hold[i - 1][j], hold[i - 1][j - 1] + prices[i]);
```

```
            }
        }
        return *max_element(not_hold[n - 1], not_hold[n - 1] + 3);
    }
};
```

Proof by Loop Invariant: `hold[i][j]` and `not_hold[i][j]` is always the maximum profit at `ith` day with `j` tranactions. `hold[i][j]` means the maximum profit you can earn at `ith` day if you hold a stock at `ith` day and have finished `j` transactions. `not_hold[i][j]` means the maximum profit you can earn at `ith` day if you do not hold a stock at `ith` day and have finished `j` transactions. Note: the whole process of buy and sell is counted as one transaction. If you only buy the stock and have not sell it, it is not a transaction.

If the above loop invariant is proof correct through the algorithm, then we could just output maximum `not_hold[n - 1][j]` cause it is the maximum profit we can earn at last day when you consider all possible transactions nunber.

Initialization:
when `i == 0` , if you buy a stock, then you need to pay `prices[0]`, and you have not sell it so it is not counted as a transation, so `hold[0][0] = -prices[0]`
, it is obvious the max profit you could earn. And if you don't buy, then of course the profit is 0, `not_hold[0][0] = 0`. `hold[0][1], hold[0][2], not_hold[0][1], not_hold[0][2]` is a negative infinity, because you cannot do one or more transactions at first day.

Maintaineance:
`kth` day with `j` transactions, there will be two cases:

1. case 1: you hold a stock at `k` day. It can be the result by 1. you buy a stock at `k` day. 2. you hold the stock at `k - 1` day, and you don't sell it at `k` day. Because `hold[k][j]` and `not_hold[k][j]` is the maximum profit at `k` day with `j` transactions, then the maximum profit is the maximum profit for case 1 is `not_hold[k - 1][j] - prices[k]`, case 2 is `hold[k - 1][j]`. The maximum profit at `k + 1` day is the larger one among the 2 cases. I.e. `hold[k][j] = max(hold[k - 1][j], not_hold[k - 1][j] - prices[k])`;. The loop invariant still holds

2. case 2: you do not hold a stock at `k` day. It can be the result by 1. you sell a stock at `k` day, then you finished a transaction. 2. you do not hold the stock at `k - 1` day, and you don't buy one at `k` day. Because `hold[k - 1][j - 1]` and `not_hold[k - 1][j]` is the maximum profit at `k` day, then the maximum profit is the maximum profit for case 1 is `hold[k - 1][j - 1] + prices[k]`, case 2 is `not_hold[k - 1][j]`. The maximum profit at `k` day is the larger one among the 2 cases. I.e. `not_hold[k][j] =`

`max(not_hold[k - 1][j], hold[k - 1][j - 1] + prices[k])`;. The loop invariant still holds.

Termination:

The algorithm terminates at when `i == n` and `j == 2`. It is holds during all the algorithm

Time complexity:
We traverse the prices vector for one time, so it is O(n) where n is the length of the prices.