309. Best Time to Buy and Sell Stock with Cooldown

Medium 3301 103 Add to List Share

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) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

Example:

```
Input: [1,2,3,0,2]
Output: 3
Explanation: transactions = [buy, sell, cooldown, buy, sell]
```

Base case:

T[-1][k][0] = T[i][0][0] = 0

T[-1][k][1] = T[i][0][1] = -Infinity

Recurrence relations:

T[i][k][0] = max(T[i-1][k][0], T[i-1][k][1] + price[i])

T[i][k][1] = max(T[i-1][k][1], T[i-1][k-1][0] - price[i])

k = +Infinity but with cooldown

There is not any difference between k and k-1.

T[i][k][0] = max(T[i-1][k][0], T[i-1][k][1] + price[i])

T[i][k][1] = max(T[i-1][k][1], T[i-1][k][0] - price[i])

We use T[i-2][k][0] in stead of T[i-1][k][0].

T[i][k][0] = max(T[i-1][k][0], T[i-1][k][1] + price[i])

T[i][k][1] = max(T[i-1][k][1], T[i-2][k][0] - price[i])

Time complexity : O(n)

Space complexity : O(1)

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        // base case
        int t_i_k_0 = 0;     // T[i][k][0] = 0
        int t_i_k_1 = INT_MIN;  // T[i][k][1] = -Infinity

        int t_i_k_0_prev = 0;

        // recurrence
        for (auto price : prices) {
            int t_i_k_0_temp = t_i_k_0;
            // T[i][k][0] = max(T[i-1][k][0], T[i-1][k][1] + prices[i])
            t_i_k_0 = max(t_i_k_0_temp, t_i_k_1 + price);
            // T[i][k][1] = max(T[i-1][k][1], T[i-2][k][0] - prices[i])
            t_i_k_1 = max(t_i_k_1, t_i_k_0_prev - price);
            // T[i - 2][k][0] = T[i - 1][k][0]
            t_i_k_0_prev = t_i_k_0_temp;
        }

        return t_i_k_0;
    }
};
```