188. Best Time to Buy and Sell Stock IV

Hard 2157 135 Add to List Share

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

Design an algorithm to find the maximum profit. You may complete at most `k` transactions.

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

Example 1:

```
Input: k = 2, prices = [2,4,1]
Output: 2
Explanation: Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.
```

Example 2:

```
Input: k = 2, prices = [3,2,6,5,0,3]
Output: 7
Explanation: Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 = 4. Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
```

Constraints:

- `0 <= k <= 100`
- `0 <= prices.length <= 1000`
- `0 <= prices[i] <= 1000`

Notation:

prices : the stock price array with length n

i : the ith day (i will go from 0 to n-1)

k : the maximum number of transactions allowed to complete

T[i][k] : the maximum profit that could be gained at the end of ith day with at most k transactions.

Base case:

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

no stock or no transaction yield no profit.

Actions:

buy : there should be 0 stock held in our hand.

sell : there should be 1 stock held in our hand.

rest

Definition:

T[i][k][0]:

the maximum profit at the end of ith day with at most k transactions and with 0 stock in our hand after taking the action.

T[i][k][1]:

the maximum profit at the end of ith day with at most k transactions and with 1 stock in our hand after taking the action.

Base case:

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

Profit is 0 if there is no stock available.

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

It's impossible to have 1 stock in hand if there is no stock.

T[i][0][0] = 0

Profit is 0 if there is no transaction are allowed.

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

It's impossible to have 1 stock in hand if there is no transaction.

Recurrence relations:

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

max (rest , sell)

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

max (rest , buy)

```cpp
class Solution {
public:
    int maxProfit(int k, vector<int>& prices) {
        if (k <= 0) {
            return 0;
        }

        // base case
        vector<int> t_i_k_0(k, 0);
        vector<int> t_i_k_1(k, INT_MIN); // T[-1][1][1]

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

        return *max_element(t_i_k_0.begin(), t_i_k_0.end());
    }
};
```