Buy & Sell Stocks

序列I型DP

121. Best Time to Buy and Sell Stock
122. Best Time to Buy and Sell Stock II
123. Best Time to Buy and Sell Stock III
188. Best Time to Buy and Sell Stock IV
309. Best Time to Buy and Sell Stock with Cooldown
714. Best Time to Buy and Sell Stock with Transaction Fee

时间序列型DP

定义:给出一个时间序列，比如股票每日价格，今天的状态取决于昨天的状态（同类型有House Robber, Paint House）

定义DP[i][j]为第i-th时间的第j种状态(比如buy/sell/cool-down)

DP[i][j] = func(DP[i-1][jj]) for jj in j (转移函数尽量画FSM理解）

时间序列型DP

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/discuss/108870/Most-consistent-ways-of-dealing-with-the-series-of-stock-problems

最多三个维度:

时间维度 i

状态维度 j

交易次数 k

set buy0 = -prices[0], sell0 = float('inf')

121. Best Time to Buy and Sell Stock

You want to maximize your profit by choosing a **single** day to buy one stock and choosing a **different** day in the future to sell that stock.

Return *the* ***maximum profit*** *you can achieve from this transaction*. If you cannot achieve any profit, return 0.

i = n; j = 2(buy/sell); k = 1(single tran)

```python
class Solution(object):
    def maxProfit(self, prices):
        # O(n)/O(n), Template but slow, j = 2 & k = 1
        dp = [[0] * 2 for _ in range(len(prices))]
        dp[0][0] = -prices[0]
        for i in range(1, len(prices)):
            dp[i][0] = max(0 - prices[i], dp[i-1][0])
            dp[i][1] = max(dp[i-1][0] + prices[i], dp[i-1][1])
        return max(dp[-1])

        # O(n)/O(n), keep a running min price
        if len(prices) ==0:
            return 0
        dp = [0] * len(prices)
        minPrice = prices[0]
        for i in range(len(prices)):
            dp[i] = max(dp[i - 1], prices[i] - minPrice)
            minPrice = min(minPrice, prices[i])
        return dp[-1]

        # compress state, O(n)/O(1)
        if len(prices) ==0:
            return 0
        profit = 0
        minPrice = prices[0]
        for i in range(len(prices)):
            profit = max(profit, prices[i] - minPrice)
            minPrice = min(minPrice, prices[i])
        return profit
```

122. Best Time to Buy and Sell Stock II

Find the maximum profit you can achieve. You may **complete as many transactions as you like** (i.e., buy one and sell one share of the stock multiple times).

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

i = n; j = 2(buy/sell); **k = inf**

```python
class Solution:
    def maxProfit(self, prices):
        # 1D DP
        n = len(prices)
        buy, sell = -prices[0], 0
        for i in range(1, n):
            buy = max(sell - prices[i], buy)
            sell = max(buy + prices[i], sell)
        return max(sell, buy)

        # state 0: buy, could come from buy or sell last round
        # state 1: sell, last round must be hold so now you can sell
        n = len(prices)
        dp = [[0] * 2 for _ in range(n)]
        dp[0][0], dp[0][1] = -prices[0], 0 # previously we cannot hold
        for i in range(1, n):
            dp[i][0] = max(dp[i-1][1] - prices[i], dp[i-1][0])
            dp[i][1] = max(dp[i-1][0] + prices[i], dp[i-1][1])
        return max(dp[-1])


        # state 0: buy, could come from buy or sell last round
        # state 1: sell, last round must be hold so now you can sell
        n = len(prices)
        dp = [[0] * 2 for _ in range(n + 1)]
        dp[0][0], dp[0][1] = float('-inf'), 0 # previously we cannot hold
        for i in range(1, n + 1):
            dp[i][0] = max(dp[i-1][1] - prices[i-1], dp[i-1][0])
            dp[i][1] = max(dp[i-1][0] + prices[i-1], dp[i-1][1])
        return max(dp[-1])


        # consider each pair of neighbors
        return sum(max(prices[i + 1] - prices[i], 0) for i in range(len(prices) - 1))
```

123. Best Time to Buy and Sell Stock III

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).

i = n; j = 2(buy/sell); **k = 2**

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # O(n)/O(1)
        # 4 states, buy1 -> sell1 -> buy2 -> sell2
        # init value: set buy to -inf, set sell to 0
        n = len(prices)
        buy1, buy2, sell1, sell2 = -prices[0], float('-inf'), float('-inf'), float('-inf')

        for i in range(1, n):
            buy1 = max(0 - prices[i], buy1) # it could be not buy i=0
            sell1 = max(buy1 + prices[i], sell1)
            buy2 = max(sell1 - prices[i], buy2)
            sell2 = max(buy2 + prices[i], sell2)

        return max(sell2, 0) # no need to complete 2 trans

        # O(n)/O(n)
        # 4 states, buy1 -> sell1 -> buy2 -> sell2
        # init value: set buy to -inf, set sell to 0
        n = len(prices)
        dp = [[0] * 4 for _ in range(n)]
        dp[0][1] = dp[0][2] = dp[0][3] = float('-inf')
        dp[0][0] = -prices[0] # buy1 for prices[0]

        for i in range(1, n):
            dp[i][0] = max(0 - prices[i], dp[i - 1][0]) # it could be not buy i=0
            dp[i][1] = max(dp[i - 1][0] + prices[i], dp[i - 1][1])
            dp[i][2] = max(dp[i - 1][1] - prices[i], dp[i - 1][2])
            dp[i][3] = max(dp[i - 1][2] + prices[i], dp[i - 1][3])

        return max(dp[-1] + [0]) # no need to complete 2 trans
```

188. Best Time to Buy and Sell Stock IV

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

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

i = n; j = 2(buy/sell); **k = k**

```python
class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        # if use n + 1, just set init buy as -inf
        # if use n, just set init buy as -prices[0]


        n = len(prices)
        if k > n // 2: # avoid TLE, make maximum number of transactions
            cur = 0
            for i in range(1, n):
                cur += max(0, prices[i] - prices[i - 1])
            return cur

        buy, sell = [float("-inf")] * (k + 1), [0] * (k + 1)
        for price in prices:
            for kk in range(1, k + 1):
                buy[kk] = max(buy[kk], sell[kk-1] - price)
                sell[kk] = max(sell[kk], buy[kk] + price)
        return sell[-1]
```

309. Best Time to Buy and Sell Stock with Cooldown

Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times) with the following restrictions:

- After you sell your stock, you cannot buy stock on the next day (i.e., cooldown one day).

i = n; j = 3(buy/sell/cooldown); k = inf

Similar with II but one more state

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        # state 0: cool down/rest
        # state 1: buy it
        # state 2: sell it

        n = len(prices)
        cooldown, buy, sell = 0, -prices[0], float('-inf')
        for i in range(1, n):
            cooldown, buy, sell = max(sell, cooldown), \
                                  max(cooldown - prices[i], buy), \
                                  buy + prices[i]
        return max(cooldown, sell)

        n = len(prices)
        dp = [[0] * 3 for _ in range(n)]
        dp[0][1] = -prices[0]
        for i in range(1, n):
            dp[i][0] = max(dp[i-1][2], dp[i-1][0])
            dp[i][1] = max(dp[i-1][0] - prices[i], dp[i-1][1])
            dp[i][2] = dp[i-1][1] + prices[i]
        # print(dp)
        return max(dp[-1])
```

714. Best Time to Buy and Sell Stock with Transaction Fee

Find the maximum profit you can achieve. You may complete as many transactions as you like, but you need to **pay the transaction fee for each transaction.**

i = n; j = 2; k = inf

Similar with II but extra fee

```python
class Solution:
    def maxProfit(self, prices: List[int], fee: int) -> int:
        n = len(prices)
        buy, sell = -prices[0], 0
        for i in range(1, n):
            buy = max(sell - prices[i], buy)
            sell = max(buy + prices[i] - fee, sell)
        return max(sell, buy)

        # state 0: buy, could come from buy or sell last round
        # state 1: sell, last round must be hold so now you can sell
        n = len(prices)
        dp = [[0] * 2 for _ in range(n)]
        dp[0][0], dp[0][1] = -prices[0], 0 # previously we cannot hold
        for i in range(1, n):
            dp[i][0] = max(dp[i-1][1] - prices[i], dp[i-1][0])
            dp[i][1] = max(dp[i-1][0] + prices[i] - fee, dp[i-1][1])
        # print(dp)
        return max(dp[-1])
```