Stock Buy and Sell – Multiple Transaction Allowed

solved by	Senan
Platform	GeeksForGeeks
difficulty	Hard
tags	Dynamic Programming
language	C++
solved on	@21/11/2024
link	https://www.geeksforgeeks.org/problems/stock-buy-and-sell2615/1
Completion	☑

Intuition

The problem is about finding the maximum profit from stock trading, given that you can buy and sell stocks with certain constraints. The goal is to decide at each day whether to buy, sell, or skip to maximize the profit.

This is a classic dynamic programming problem because the decision on one day affects the options for subsequent days, making it necessary to consider overlapping subproblems and optimal substructures.

Approach

1. **State Representation:**
   Define a recursive function `f(i, buy)` where:
   - `i` represents the current day (index in the `prices` array).
   - `buy` is a binary indicator:
     - `1` if you are allowed to buy on this day.
     - `0` if you are allowed to sell on this day.
2. **Base Case:**
   If `i == prices.size()`, we have processed all days, and the profit is `0` since no further transactions are possible.
3. **Recursive Transition:**
   - If `buy == 1` (allowed to buy):
     - **Buy today**: Subtract the price of the stock today and transition to the next day with `buy = 0`.
     - **Don't buy**: Skip buying today and transition to the next day with `buy = 1`.
     - Take the maximum of these two options.
   - If `buy == 0` (allowed to sell):
     - **Sell today**: Add the price of the stock today and transition to the next day with `buy = 1`.
     - **Don't sell**: Skip selling today and transition to the next day with `buy = 0`.
     - Take the maximum of these two options.

4. **Memoization:**
   Use a
   `dp` array (`dp[i][buy]`) to store results of previously computed states to avoid redundant calculations.
5. **Initialization:**
   Start the recursion from day
   `0` with `buy = 1` (allowed to buy initially).
6. **Final Answer:**
   The answer is the result of the recursive function starting from
   `f(0, 1)`.

Complexity

Time Complexity:

- **Recursive Calls:** There are `n` days, and for each day, we have two states (`buy = 0 or 1`). Hence, the total number of states is `O(n * 2)`.
- Each state is computed only once due to memoization.
- **Total Time Complexity:** `O(n)`.

Space Complexity:

- **Memoization Table:** `O(n * 2)`, since we store results for `n` days and 2 possible `buy` states.
- **Recursive Stack:** `O(n)` in the worst case (depth of recursion is equal to the number of days).
- **Total Space Complexity:** `O(n)`.

Code

```
class Solution {
    int f(int i, int buy, vector<int>& prices, vector<vector<int>> &dp) {
        if (i == prices.size()) return 0;

        if (dp[i][buy] != -1) return dp[i][buy];

        if (buy) {
            int buyToday = -prices[i] + f(i + 1, 0, prices, dp);
            int dontBuy = f(i + 1, 1, prices, dp);
            return dp[i][buy] = max(buyToday, dontBuy);
        } else {
            int sellToday = prices[i] + f(i + 1, 1, prices, dp);
            int dontSell = f(i + 1, 0, prices, dp);
            return dp[i][buy] = max(sellToday, dontSell);
        }
    }
public:
    int maximumProfit(vector<int>& prices) {
        vector<vector<int>> dp(prices.size(), vector<int>(2, -1));
        return f(0, 1, prices, dp);
    }
};
```