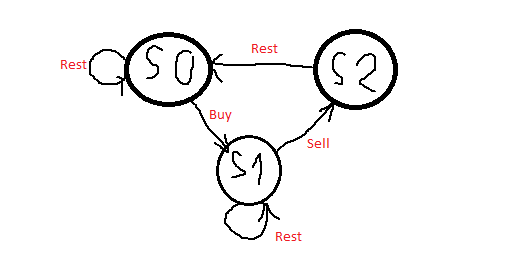
Hi,

I just come across this problem, and it's very frustating since I'm bad at DP.

So I just draw all the actions that can be done.

Here is the drawing (Feel like an elementary ...)



[Original image](https://imgur.com/wvR4TN8)

There are three states, according to the action that you can take.

Hence, from there, you can now the profit at a state at time i as:

s0[i] = max(s0[i - 1], s2[i - 1]); // Stay at s0, or rest from s2

s1[i] = max(s1[i - 1], s0[i - 1] - prices[i]); // Stay at s1, or buy from s0

s2[i] = s1[i - 1] + prices[i]; // Only one way from s1

Then, you just find the maximum of s0[n] and s2[n], since they will be the maximum profit we need (No one can buy stock and left with more profit that sell right :) )

Define base case:

s0[0] = 0; // At the start, you don't have any stock if you just rest

s1[0] = -prices[0]; // After buy, you should have -prices[0] profit. Be positive!

s2[0] = INT\_MIN; // Lower base case

Here is the code :D

class Solution {

public:

int maxProfit(vector<int>& prices){

if (prices.size() <= 1) return 0;

vector<int> s0(prices.size(), 0);

vector<int> s1(prices.size(), 0);

vector<int> s2(prices.size(), 0);

s1[0] = -prices[0];

s0[0] = 0;

s2[0] = INT\_MIN;

for (int i = 1; i < prices.size(); i++) {

s0[i] = max(s0[i - 1], s2[i - 1]);

s1[i] = max(s1[i - 1], s0[i - 1] - prices[i]);

s2[i] = s1[i - 1] + prices[i];

}

return max(s0[prices.size() - 1], s2[prices.size() - 1]);

}

};