/\*

Say you have an array for which the ith 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:

prices = [1, 2, 3, 0, 2]

maxProfit = 3

transactions = [buy, sell, cooldown, buy, sell]

股票买卖要求每次交易间隔一天，和股票买卖4一样，动态规划找两个关系。

local[i]=max(global[i-3]+max(diff,0),local[i-1]+diff);

global[i]=max(global[i-1],local[i]);

\*/

class Solution {

public:

int maxProfit(vector<int>& prices)

{

int n=prices.size();

if(n<2)

return 0;

vector<int> global(n,0);

vector<int> local(n,0);

for(int i=1;i<n;i++)

{

int diff=prices[i]-prices[i-1];

if(i>3)

local[i]=max(global[i-3]+max(diff,0),local[i-1]+diff);

else

local[i]=max(max(diff,0),local[i-1]+diff);

global[i]=max(global[i-1],local[i]);

}

return global[n-1];

}

};