/\*

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 at most k transactions.

Note:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

题目的关键是要写出下面的动态转移方程:

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

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

这道题是Best Time to Buy and Sell Stock的扩展，现在我们最多可以进行两次交易。我们仍然使用动态规划来完成，事实上可以解决非常通用的情况，也就是最多进行k次交易的情况。

这里我们先解释最多可以进行k次交易的算法，然后最多进行两次我们只需要把k取成2即可。我们还是使用“局部最优和全局最优解法”。我们维护两种量，一个是当前到达第i天可以最多进行j次交易，最好的利润是多少（global[i][j]），另一个是当前到达第i天，最多可进行j次交易，并且最后一次交易在当天卖出的最好的利润是多少（local[i][j]）。下面我们来看递推式。

全局的比较简单，

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

也就是去当前局部最好的，和过往全局最好的中大的那个（因为最后一次交易如果包含当前天一定在局部最好的里面，否则一定在过往全局最优的里面）。

对于局部变量的维护，递推式是

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

也就是看两个量，第一个是全局到i-1天进行j-1次交易，然后加上今天的交易，如果今天是赚钱的话（也就是前面只要j-1次交易，最后一次交易取当前天,当天买卖），第二个量则是取local第i-1天j次交易，然后加上今天的差值（这里因为local[i-1][j]比如包含第i-1天卖出的交易，所以现在变成第i天卖出，并不会增加交易次数，而且这里无论diff是不是大于0都一定要加上，因为否则就不满足local[i][j]必须在最后一天卖出的条件了）。

上面的算法中对于天数需要一次扫描，而每次要对交易次数进行递推式求解，所以时间复杂度是O(n\*k)，如果是最多进行两次交易，那么复杂度还是O(n)。空间上只需要维护当天数据皆可以，所以是O(k)，当k=2，则是O(1)。"

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

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

\*/

class Solution {

public:

int maxProfit(int k, vector<int>& prices)

{

if(prices.size()<2)

return 0;

if(k>=prices.size())

{

int ret=0;

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

if(prices[i]>prices[i-1])

ret+=prices[i]-prices[i-1];

return ret;

}

else if(k<=0)

return 0;

vector<vector<int>> local;

vector<vector<int>> global;

for(int i=0;i<prices.size();i++)

{

vector<int> m1(k+1,0);

local.push\_back(m1);

global.push\_back(m1);

}

for(int j=1;j<=k;j++)

{

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

{

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

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

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

}

}

return global[prices.size()-1][k];

}

};