这道题简直是经典的动态规划问题的解法。做了十几道动态规划以后，发觉动态规划这玩意，最常用的两个函数就是min()和max()。这道题的状态分析图简直是经典的没道理。能解决这一类问题。

空间复杂度比较高。但是时间上beats了99.6的人。很不错的代码。

## 问题描述：

Say you have an array for which the *i*th 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:**

**Input:** [1,2,3,0,2]

**Output:** 3

**Explanation:** transactions = [buy, sell, cooldown, buy, sell]

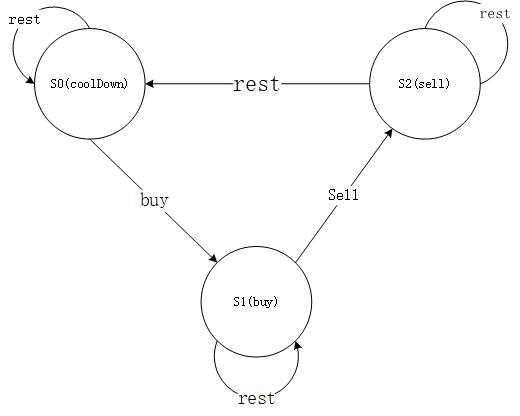
## 解题思路：

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 ...)



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

|  |  |
| --- | --- |
|  |  |

左侧的代码是运行时间比较慢的，右侧的运行时间是快的，通过这个代码我们可以看到维护数组的代价还是低于计算耗费的时间的。尽量减少计算。