剑指42 连续子数组的最大和

完全没想出。

法一：暴力穷举，O(n^2)
法二：分治思想，O(nlogn)，还没研究过。
法三：动态规划，O(n)

重点：动态规划的状态转移方程如何思考想到。
解析：设动态规划列表dp，dp[i]代表以元素nums[i]为结尾的连续子数组最大和，注意必须是包含nums[i]的。（考虑动态规划的划分子问题分治法，于是直接运用容易想到dp[i]，其中i代表长度i的整个字符串，然后dp[i]等于它的左右-1子串的dp的等等，然而这种思路是错误的，考虑两边边界什么的，把问题复杂化，往深去思考直接出不来了。）为什么必须包含，因为我们要在满足连续子数组要求的基础上考虑dp的递推性，即dp[i + 1]和dp[i]之间要有递推的某种关联。如果dp[i - 1] <= 0,说明dp[i - 1]对dp[i]产生负贡献，还不如直接选nums[i]本身。

转移方程：

nums	-2	1	-3	4	-1	2	1	-5	4
dp	-2	1	-2	4	3	5	6	1	5

状态定义：

$dp[i]$ 代表以元素 $nums[i]$ 为结尾的连续子数组最大和

转移方程：

$$dp[i] = \begin{cases} dp[i-1] + nums[i] & , dp[i-1] > 0 \\ nums[i] & , dp[i-1] \le 0 \end{cases}$$

这个动态规划的转移方程定义思路要注意，最终的结果并不是递推到最后的dp[n]的值，而是是从n个dp[]中选择最大的一个值。逻辑是自洽的，即最终结果必然是以某个元素x作为结尾。

```
// 看了思路后我的题解：
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int max = nums[0];
        int dpsum = max;
```

```
        // 利用一个变量dpsum来维护对于当前i的dp[i-1]的值来实现O(1)空间复杂度，有点【滚动数组】的思想
        for(int i = 1; i < nums.size(); i++) {
            if (dpsum > 0) {
                dpsum = dpsum + nums[i];
            }
            else {
                dpsum = nums[i];
            }
            max = (max > dpsum) ? max : dpsum;
        }
        return max;
    }
};
```