动态规划 简单题

剑指 Offer 42. 连续子数组的最大和

输入一个整型数组，数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值。

要求时间复杂度为O(n)。

输入: nums = [-2,1,-3,4,-1,2,1,-5,4]

输出: 6

解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

Java

```java
class Solution {
    public int maxSubArray(int[] nums) {
        // 这题一看就是动态规划啊
        int[] dp = new int[nums.length];
        dp[0] = nums[0];
        int max = dp[0];
        // 因为要考虑是否将上一个元素加到当前子序列中，所以用一个max值来判断
        for(int i = 1; i < dp.length; i++){
            // 判断当前元素加入到子序列与当前元素相比较
            dp[i] = Math.max(dp[i-1] + nums[i], nums[i]);
            // 第i个元素所拥有的最长子序列和不一定就是最大的，所以这里要做标记
            max = Math.max(max, dp[i]);
        }
        return max;
    }
}
```

53. 最大子序和4

本题与53. 最大子序和相同，这里也进行描述。

给定一个整数数组 `nums` ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

输入：nums = [-2,1,-3,4,-1,2,1,-5,4]**输出：**6**解释：**连续子数组 [4,-1,2,1] 的和最大，为 6 。

输入：nums = [-100000]**输出：**-100000

解答与上一道题目一致。时间空间均为O(N)

OpenGL Shading Language

```
class Solution {
    public int maxSubArray(int[] nums) {
        // 题目要求最少包含一个元素
        if(nums.length == 1) return nums[0];
        int[] dp = new int[nums.leth];
        dp[0] = nums[0];
        int max = dp[0]; // 这里是否可以初始化为0？不行，这里遍历是从第二个元素开始的
        for(int i = 1; i < dp.length; i++) {
            dp[i] = Math.max(dp[i-1] + nums[i], nums[i]);
            max = Math.max(max, dp[i]);
        }
        return max;
    }
}
```

★152. 乘积最大子数组

给你一个整数数组 `nums` ，请你找出数组中乘积最大的连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。

输入: [2,3,-2,4]

输出: 6

解释: 子数组 [2,3] 有最大乘积 6。

输入: [-2,0,-1]

输出: 0

解释: 结果不能为 2, 因为 [-2,-1] 不是子数组。

OpenGL Shading Language

```
class Solution {
    public int maxProduct(int[] nums) {
        // 和53题、剑指offer42题目类似，从和变为乘积
        // 这里也是要求最少有一个数字
        // 跟前面的题有所不同，负负得正，ds可能会逆袭，这里还需要记录最小值
        if(nums.length == 1) return nums[0];
        int max = Integer.MIN_VALUE, imax = 1, imin = 1; // 这里初始化为1，保证第一个相乘时的正确性

        for(int i = 0;i < nums.length; i++){
            // 如果当前是负数，[-2, 3, -4]，当你遍历到3时，
            // imax = -2 * 3 = -6 --> 3,
            // imin = -2 * 3 = -6 --> -6
            // 记住最小值，遇到下一个负数时可能会反转
            // 而第一个负数，为什么要反转呢，拿一个较小的正数去乘于负数得到比较小的负数作为imax
            // 而更小的负数作为 imin
            if(nums[i] < 0) {
                int tmp = imax;
                imax = imin;
                imin = tmp;
            }
            imax = Math.max(imax * nums[i], nums[i]);
            imin = Math.min(imin * nums[i], nums[i]);
            max = Math.max(max, imax);
        }
        return max;
    }
}
```