152. Maximum Product Subarray

link

`[2, -5, -2, -4, 3]`我的思路最大值就是所有非的乘起来, 如果大于0 直接return, 小于0的话从头开始除直到第一个负数位置

```
public int maxProduct(int[] nums) {
    int start = -1, i = 0, max = Integer.MIN_VALUE;
    while(i < nums.length){
        while(i < nums.length && nums[i] != 0){
            if(start == -1)
                start = i;
            i++;
        }
        max = Math.max(max, getMaxValue(nums, start, i));
        if(i == nums.length) break;
        if(nums[i] == 0) max = Math.max(max, 0);
        start = -1;
        i++;
    }
    return max;
}
public int getMaxValue(int[] nums, int start, int end){
    if(start == -1) return 0;
    int res = nums[start];
    if(start == end - 1) return res;
    int i = 0;
    for(i = start + 1; i < end; ++i){
        res *= nums[i];
    }
    if(res > 0) return res;

    int left = res;
    i = end - 1;
    while(i >= start && nums[i] > 0){
        left /= nums[i];
        --i;
    }
    left /= nums[i];
    int right = res;
    i = start;
    while(i < end && nums[i] > 0){
        right /= nums[i];
        ++i;
    }
    right /= nums[i];
    return Math.max(left, right);
}
```

一样的思路, 但是更好的写法, 用product记录, 遇到0就把product置1

```
public int maxProduct(int[] nums) {
    int max = Integer.MIN_VALUE, product = 1;
    int len = nums.length;
    for(int i = 0; i < len; i++) {
        max = Math.max(product *= nums[i], max);
         if (nums[i] == 0) product = 1;
    }
     product = 1;
     for(int i = len - 1; i >= 0; i--) {
         max = Math.max(product *= nums[i], max);
         if (nums[i] == 0) product = 1;
     }
return max;
```

动态规划思路这道题妙就妙在它不仅仅依赖了一个状态（前一个数所能获得的最大乘积），而是两个状态（最大和最小乘积）。比较简单的dp问题可能就只是会建立一个dp[]，然后把最大值放到其中。但是这道题给我们打开了新的思路：我们的dp数组里面可以存更多的信息。而上面的解法之所以没有用dp数组的原因是dp[i]只依赖于dp[i - 1]因此没有必要把前面所有的信息都存起来，只需要存前一个dp[i-1]的最大和最小的乘积就可以了。下面的代码使用了自定义的内部类Tuple,从而可以同时存imax和imin,并将所有的imax和imin存到了dp数组中。虽然稍显复杂，但是有助于加深理解。

```
int maxProduct(int A[], int n) {
    // store the result that is the max we have found so far
    int r = A[0];

    // imax/imin stores the max/min product of
    // subarray that ends with the current number A[i]
    for (int i = 1, imax = r, imin = r; i < n; i++) {
        // multiplied by a negative makes big number smaller, small number bigger
        // so we redefine the extremums by swapping them
        if (A[i] < 0)
            swap(imax, imin);

        // max/min product for the current number is either the current number
    itself
        // or the max/min by the previous number times the current one
        imax = max(A[i], imax * A[i]);
        imin = min(A[i], imin * A[i]);

        // the newly computed max value is a candidate for our global result
        r = max(r, imax);
    }
    return r;
}
```