栈与队列

150. 逆波兰表达式求值

简单 5min

题目

给你一个字符串数组 `tokens` ，表示一个根据 逆波兰表示法 表示的算术表达式。

请你计算该表达式。返回一个表示表达式值的整数。

注意：

- 有效的算符为 `'+'`、`'-'`、`'*'` 和 `'/'` 。
- 每个操作数（运算对象）都可以是一个整数或者另一个表达式。
- 两个整数之间的除法总是 **向零截断** 。
- 表达式中不含除零运算。
- 输入是一个根据逆波兰表示法表示的算术表达式。
- 答案及所有中间计算结果可以用 **32 位** 整数表示。

示例 1：

```
输入：tokens = ["2","1","+","3","*"]
输出：9
解释：该算式转化为常见的中缀算术表达式为：((2 + 1) * 3) = 9
```

示例 2：

```
输入：tokens = ["4","13","5","/","+"]
输出：6
解释：该算式转化为常见的中缀算术表达式为：(4 + (13 / 5)) = 6
```

示例 3：

```
输入：tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
输出：22
解释：该算式转化为常见的中缀算术表达式为：
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

提示：

- `1 <= tokens.length <= 104`

- `tokens[i]` 是一个算符（`"+"`、`"-"`、`"*"` 或 `"/"`），或是在范围 `[-200, 200]` 内的一个整数

逆波兰表达式：

逆波兰表达式是一种后缀表达式，所谓后缀就是指算符写在后面。

- 平常使用的算式则是一种中缀表达式，如 `( 1 + 2 ) * ( 3 + 4 )` 。
- 该算式的逆波兰表达式写法为 `( ( 1 2 + ) ( 3 4 + ) * )` 。

逆波兰表达式主要有以下两个优点：

- 去掉括号后表达式无歧义，上式即便写成 `1 2 + 3 4 + *` 也可以依据次序计算出正确结果。
- 适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中

题解

简单的栈

```cpp
class Solution {
public:
    int evalRPN(vector<string>& tokens) {
        stack<long long>po;//要求是32位
        int len=tokens.size();
        for(string x:tokens){
            if(x=="+"){
                long long a=po.top();
                po.pop();
                long long b=po.top();
                po.pop();
                po.push(a+b);
            }
            else if(x=="-"){
                long long a=po.top();
                po.pop();
                long long b=po.top();
                po.pop();
                po.push(b-a);//注意顺序是b-a
            }
            else if(x=="*"){
                long long a=po.top();
                po.pop();
                long long b=po.top();
                po.pop();
                po.push(a*b);
            }
            else if(x=="/"){
                long long a=po.top();
                po.pop();
                long long b=po.top();
                po.pop();
                po.push(b/a);//注意顺序
            }

            else po.push(stoi(x));//将字符串转化为int
            cout<<po.top()<<endl;//debug
        }
        return po.top();
    }
};
```

注意事项：

- 比对字符串要用双引号
- `stoi()` 用来把字符串转为int值

- 32位整数要用long long

239. 滑动窗口最大值

困难

题目

力扣题目链接

给你一个整数数组 `nums`，有一个大小为 `k` 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 `k` 个数字。滑动窗口每次只向右移动一位。

返回 *滑动窗口中的最大值* 。

示例 1：

```
输入：nums = [1,3,-1,-3,5,3,6,7], k = 3
输出：[3,3,5,5,6,7]
解释：
滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

示例 2：

```
输入：nums = [1], k = 1
输出：[1]
```

提示：

- `1 <= nums.length <= 105`
- `-104 <= nums[i] <= 104`
- `1 <= k <= nums.length`

题解

这是使用单调队列的经典题目。

粗略看了一下暴力一下不就得了，实际执行会爆内存。

暴力方法，遍历一遍的过程中每次从窗口中再找到最大的数值，这样很明显是O(n × k)的算法。

因此考虑使用单调队列的方法，和之前的单调栈很相似，都是保持递增的状态，因为小的数据已经没有用了，同时又因为是滑动窗口，所以在左侧要把超出范围的也去掉，因此单调队列=单调栈+滑动窗口

```
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
```

```cpp
        vector<int> ans;
        deque<int> q; // 双端队列，两边都能编辑
        for (int i = 0; i < nums.size(); i++) {
            // 1. 入
            while (!q.empty() && nums[q.back()] <= nums[i]) {
                q.pop_back(); // 维护 q 的单调性，如果当前的数比队列里的大，队列里的就没啥用，全部裁员
            }
            q.push_back(i); // 加入新员工
            // 2. 出
            if (i - q.front() >= k) { // 队首已经离开窗口了
                q.pop_front();
            }
            // 3. 记录答案
            if (i >= k - 1) {//窗口大小够了
                // 由于队首到队尾单调递减，所以窗口最大值就是队首
                ans.push_back(nums[q.front()]);
            }
        }
        return ans;
    }
};

```