题目原址-239

参考

- 小美图解剑指Offer59题 [Java]
- 花花酱 LeetCode 239 [C++]
  - video
  - code
- 暴力法 [C++]
- 单调队列解题详解

题目描述

给定一个数组 `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。

要求:返回滑动窗口中的最大值。

示例

```
输入: nums = [1,3,-1,-3,5,3,6,7], 和 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
```

提示

- $1 <= nums.length <= 10^5$
- $-10^4 <= nums[i] <= 10^4$
- $1 <= k <= nums.length$

进阶

你能在线性时间复杂度内解决此题吗?

初步分析

输出的数组大小为$nums.lengths - k + 1$

参考方法

Brute Force

扫描每个滑动窗口的所有数字并找出其中的最大值.

代码

粗略代码

> 不涉及代码的格式等细节

```
int max (vector<int> win){
    int m = win[0];
    for(auto i ; win){
        if(i > m) m = i;
    }
    return m;
}

vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    vector<int> win{};
    vector<int> maxSW{};

    for(auto i = 0; i < nums.lengths-k; i++){
        for(auto j = i, m = 0; j < i + k; j++, m++)
            win[m] = nums[j];

        maxSW[i] = max(win);
    }
}
```

错误分析

改版1

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

        for (auto i = 0; i < nums.size() - k; i++)
        {
            for (auto j = i, m = 0; j < i + k; j++, m++)
                win[m] = nums[j];

            maxSW[i] = max(win);
        }

        return maxSW;
    }
```

```
    int max(vector<int> win)
    {
        int m = win[0];
        for (auto& i : win)
            if (i > m)
                m = i;

        return m;
    }
};
```

错误提示

```
Line 1034: Char 9: runtime error: reference binding to null pointer of type 'int'
(stl_vector.h)
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior
/usr/bin/../lib/gcc/x86_64-linux-
gnu/9/../../../../include/c++/9/bits/stl_vector.h:1043:9
```

错误分析

正确

```
// Philos
class Solution
{
public:
    vector<int> maxSlidingWindow(vector<int> &nums, int k){
        vector<int> ans{};

        for (auto i = 0; i != nums.size() - k + 1; i++){
            int maxSW = nums[i]; // 默认窗口首元素最大
            for (auto j = i; j != i + k; j++)
                if(nums[j] > maxSW)
                    maxSW = nums[j];

            ans.push_back(maxSW);
        }

        return ans;
    }
};
```

leetcode测试超时

```cpp
// 花花酱
class Solution {
public:
  vector<int> maxSlidingWindow(vector<int>& nums, int k) {
    vector<int> ans;

    for (int i = 0; i != (nums.size()-k)+1; ++i) { // n-k+1
      ans.push_back(*max_element(nums.begin() + i, nums.begin() + i + k));
    }

    return ans;
  }
};
```

max_element

时间/空间复杂度分析

二级循环:$T(n) = (n-k+1)k$

没有辅助空间

BST/Multiset(等复习完,再看)

```cpp
// Author: Huahua
class Solution
{
public:
    vector<int> maxSlidingWindow(vector<int> &nums, int k)
    {
        vector<int> ans;

        if (nums.empty())
            return ans;

        multiset<int> window(nums.begin(), nums.begin() + k - 1);
        for (int i = k - 1; i < nums.size(); ++i)
        {
            window.insert(nums[i]);
            ans.push_back(*window.rbegin());
            if (i - k + 1 >= 0)
                window.erase(window.equal_range(nums[i - k + 1]).first);
        }

        return ans;
    }
};
```

Monotonic Queue/Deque

Huahua

Monotonic Queue:左大右小的双端队列

```
class MonotonicQueue{
public:
    void push(int e); // push an element on the queue,then will pop all elements smaller than e
    void pop(); // pop the first(max) element
    int max() const; // get the max element
}
```

```
  滑动窗口的位置                Monotonic Queue       最大值
---------------                 ---------------       -----
[1]  3  -1 -3  5  3  6  7     [1]                     -     push 1,1即为max
[1  3]  -1 -3  5  3  6  7     [3]                     -     push 3, 3 > 1, 1出队
[1  3  -1] -3  5  3  6  7     [3,-1]                  3     push -1, -1 < 3, -1入队; 3个元素了, 最大值3
 1 [3  -1  -3] 5  3  6  7     [3,-1,-3]               3     窗口移位1, push -3, -3 < -1, -3入队; 最大值3
 1  3 [-1  -3  5] 3  6  7     [5]                     5     窗口移位1, push 5, 5 > -1; -1,-3出队; 最大值5
 1  3  -1 [-3  5  3] 6  7     [5,3]                   5     窗口移位1, push 3, 3 < 5, 3入队; 最大值5
 1  3  -1  -3 [5  3  6] 7     [6]                     6     窗口移位1, push 6, 6 > 5, 5出队; 最大值5
 1  3  -1  -3  5 [3  6  7]    [7]                     7     窗口移位1, push 7, 7 > 6, 6出队; 最大值7
```

> 注:队尾元素可能后面窗口的最大值

代码

```
// Author: Huahua
class MonotonicQueue{
public:
    void push(int e){
        while (!data_.empty() && e > data_.back()) // 出队比e小的元素,然后入队
            data_.pop_back();

        data_.push_back(e);
    }

    void pop(){ // pop最大的元素
        data_.pop_front();
    }

    int max() const { return data_.front(); } // get最大的元素

private:
    deque<int> data_; // 支持随机访问的双端队列
};

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

```
        MonotonicQueue q;
        vector<int> ans;

        for (int i = 0; i < nums.size(); ++i){
            q.push(nums[i]);
            if (i - k + 1 >= 0){ // 有足够的元素(k个)了
                ans.push_back(q.max()); // 取最大值加入ans中

                if (nums[i - k + 1] == q.max()) // 如果当前元素(...[...]i...)与窗口的当前最大元素相等时 (与视频理解有点不一样)
                    q.pop();
            }
        }
        return ans;
    }
};
```

时间/空间复杂度分析

时间:

优化(待分析)

```
// Author: Huahua
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque<int> index;
        vector<int> ans;

        for(int i = 0; i != nums.size(); ++i){
            for(;!index.empty()  && nums[i] >= nums [index.back()];)
                index.pop_back();

            index.push_back(i);

            if(i - k + 1 >= 0)
                ans.push_back(nums[index.front()]);

            if(i - k + 1 >= index.front())
                index.pop_front();
        }

        return ans;
    }
};
```

> LeetCode测试时间没有明显减少

labuladong