



程序设计思维与实践

Thinking and Practice in Programming

数学基础与方法应用 | 内容负责：魏安然/李子晗



1

数 学 基 础

Prefix sum and Difference

取模运算

- 取模运算(mod), 也即 C/C++ 中的 “%”
- $a \bmod p = b$, b 是 a 除以 p 的余数
- 读作 “ a 模 p 等于 b ”
- 为了方便大家观看, 就直接用 “%” 代替 mod 表示
- 任何一个整数 a , 对于一个确定的模数 p , 都可以转换为如下形式

$$a = kp + b$$

$$0 < b < p$$

$$k \in \mathbb{Z}$$

而此时的 b , 也就等于 $a \bmod p$, 而 k 就是 $\left\lfloor \frac{a}{p} \right\rfloor$

取模运算

- 取模运算有一些运算性质
- $(a \pm b) \% p = (a \% p \pm b \% p) \% p$

$$a = k_1 p + a_0$$

$$b = k_2 p + b_0$$

$$(a \pm b) \% p = ((k_1 p + a_0) \pm (k_2 p + b_0)) \% p = (a_0 \pm b_0) \% p$$

- $(a \times b) \% p = ((a \% p) \times (b \% p)) \% p$

$$(a \times b) \% p = ((k_1 p + a_0) \times (k_2 p + b_0)) \% p = (a_0 \times b_0) \% p$$

- 注意, $\frac{a}{b} \% p \neq \frac{a \% p}{b \% p} \% p$
- 根据费马小定理, $\frac{1}{a} = a^{p-2}$

取模运算

- 有些题目中会涉及到取模运算，这种题目往往会出现直接乘中间过程爆int或者long long的情况（溢出），而取模的结果不会爆int或者long long
- 比如计算 $(10^{15} \times 10^{16}) \% (10^9 + 7)$ ，如果我们直接计算 $10^{15} \times 10^{16}$ ，则会爆long long，但是如果使用取模运算的运算规则，则不会出现爆long long的情况。
- 根据计算规则，也就衍生出常用的 C/C++ 写法
- $a += b, a \% = p \rightarrow a = (a + b) \% p$
- $a *= b, a \% = p \rightarrow a = (a * b) \% p$
- 在编写程序时，一定要注意结果溢出的问题

位运算

- $\&$ 按位与
- $|$ 按位或
- \sim 取反
- \wedge 按位异或
- $>>$ 右移
- $<<$ 左移

位运算

- $1 \ll n$ 计算 2^n
- $a \ll 1$ 计算 $a \times 2$ $a \gg 1$ 计算 $a/2$ (向下取整)
- $a \ll m$ $a \gg m$
- $a \& 1$ 若为真, 则 a 为奇数, 否则为偶数
- $a \& (1 \ll m)$
- $a \oplus a \oplus 1$
- $-a = \sim a + 1$

快速幂

- 计算 $a^b \bmod p$ 的结果 $0 < a, b, p \leq 10^9$
- 如果直接循环计算，则需要 $O(b)$ 的复杂度，会TLE
- 引入分治的思想
- $$a^b = \begin{cases} (a^{\frac{b}{2}})^2, & b \text{ 是偶数} \\ a(a^{\lfloor \frac{b}{2} \rfloor})^2, & b \text{ 是奇数} \end{cases}$$
- 每次可以将 b 除以 2，复杂度 $O(\log b)$

快速幂

```
LL qpow(LL a, LL b, LL p)
{
    if(!b) return 1;
    LL now = qpow(a, b/2, p);
    if(b&1) return a*now%p*now%p;
    else now*now%p;
}
```

```
LL qpow(LL a, LL b, LL p)
{
    LL ans = 1;
    for(a%=p; b; a=a*a%p, b>>=1) if(b&1) ans=ans*a%p;
    return ans;
}
```

快速幂

- 计算 $a \times b \bmod p$ 的结果 $0 < a, b, p \leq 10^{18}$
- $(10^{15} \times 10^{16}) \% (10^{17} + 7)$
- 直接计算会爆long long, 用取模运算规则计算也还是会爆long long
- 像快速幂那样把乘法展开

快速幂

- 计算 A^b 的结果, A 是一个方阵
- 因为矩阵乘法满足结合律, 所以这个也可以像快速幂那样运算



前缀和与差分

Prefix sum and Difference

前缀和

- 前缀和作用
 - $O(1)$ 求出一个区域所有元素数值之和
 - 当涉及快速求取某一区域和时，可考虑使用前缀和进行快速计算
 - 即前缀和通常用于优化算法中的某一步骤，进而降低复杂度
- 一维前缀和
 - $\text{sum}[i] = \text{sum}[i-1] + a[i]$
 - $\text{sum}[L, R] = \text{sum}[R] - \text{sum}[L-1]$

二维前缀和

- 二维前缀和涉及基础容斥
- 预处理公式
 - $\text{sum}[x, y] = \text{sum}[x-1, y] + \text{sum}[x, y-1] - \text{sum}[x-1, y-1] + a[x, y]$
- 快速计算区域和
 - 左上角 (a, b)
 - 右下角 (c, d)
- $\text{SUM} = \text{sum}[c, d] - \text{sum}[a-1, d] - \text{sum}[c, b-1] + \text{sum}[a-1, b-1]$

	1						b			d
1										
a										
c										

差分

- 差分构造方式
 - 原数组 A , 差分数组 B , 数组范围 $[1, n]$
 - $B[1] = A[1] - A[0] = A[1]$
 - $B[i] = A[i] - A[i-1]$
- 差分特点
 - B 数组前缀和 $\Leftrightarrow A$ 数组元素值
 - $\text{SUM}\{B[1 \sim i]\} = A[i]$
 - A 数组的区间加 $\Leftrightarrow B$ 数组的单点修改
 - $A[L] \sim A[R]$ 均加上 c
 - 等价于 $B[L] += c, B[R+1] -= c$



例 1 讲 解

E x a m p l e 4

例1讲解

- 题意
 - 长度为 n 的数组，一共 q 次操作， $1 \leq n, q \leq 10^5$
 - 每次操作给出 L, R, c ，表示区间 $[L, R]$ 中各个数均加上 c
 - 求 q 次操作结束后，数组中各个元素值？
- 思路
 - 暴力做法， $O(qn)$ ，超时
 - 将原数组转变为差分数组，将区间修改转变为单点修改
 - $A[L, R] += c \Leftrightarrow B[L] += c, B[R+1] -= c$
 - B 数组前缀和即为 A 数组最终数值， $O(n + q)$



4

尺取法

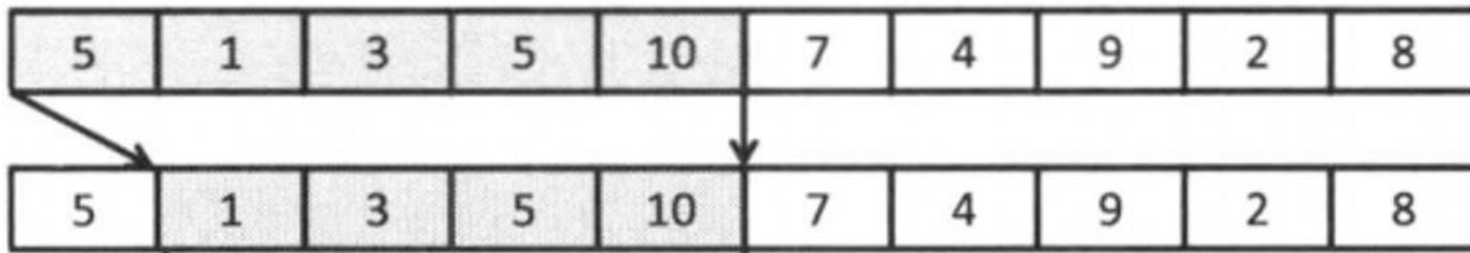
Two Pointers

尺取法

- 概念讲解
 - 尺取法，又称双指针法，是数组上的一种常见操作
 - 以下述经典例题为例进行讲解
- 题意
 - 长度为 n 的数组，每个数均为正整数
 - 给出正整数 S ，要求 $O(n)$ 内求出长度最小的连续区间，使 $S \leq \text{区间和}$
- 样例
 - $[1\ 2\ 3\ 4\ 5]$ ， $S = 11$
 - $[1\ 2\ 3\ 4\ 5]$ ， $11 \leq 3 + 4 + 5$ ，答案为 3

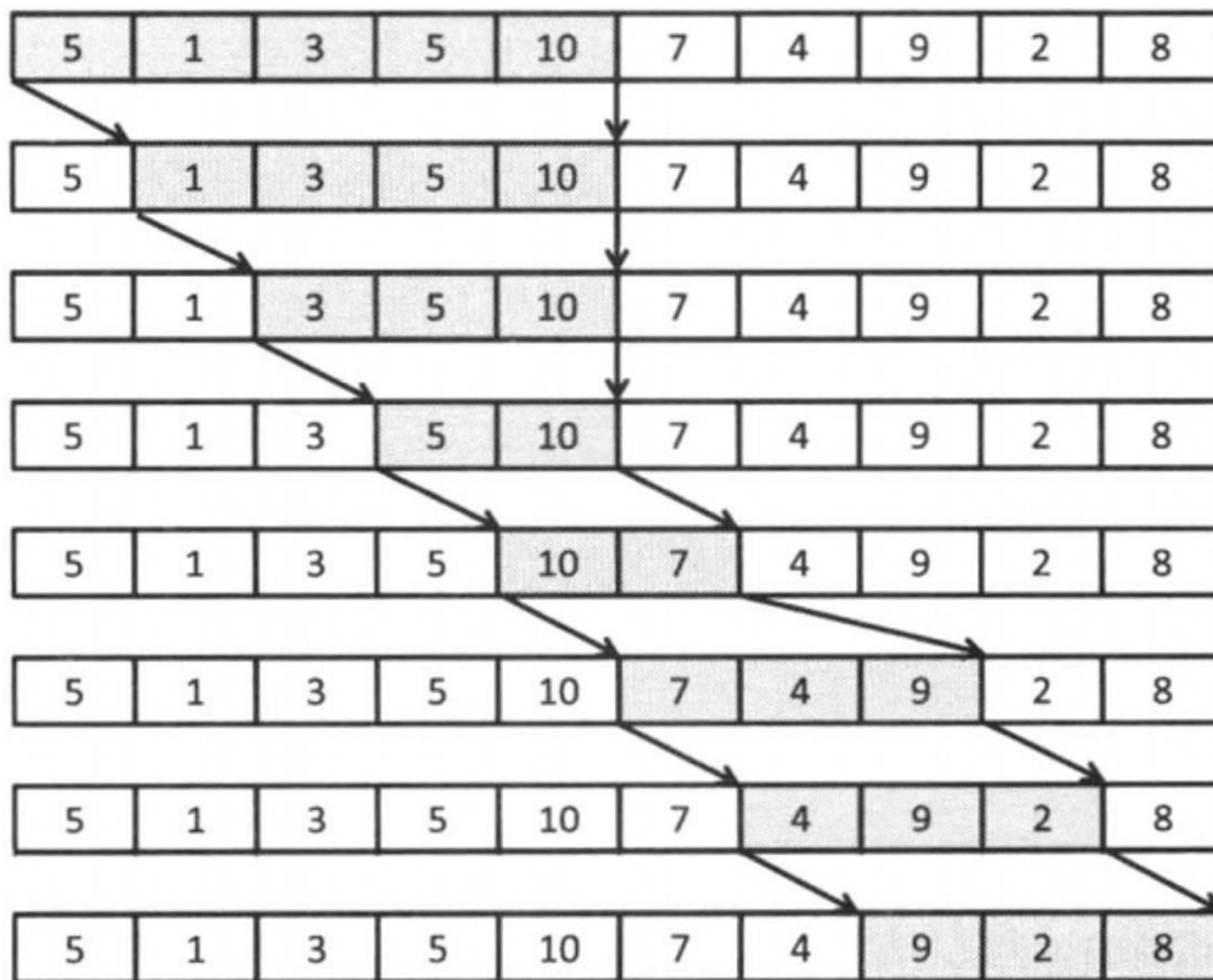
尺取法

- 以 $[5\ 1\ 3\ 5\ 10\ 7\ 4\ 9\ 2\ 8]$, $S = 15$ 为例介绍该算法
- 具体流程
 - 维护双指针 L 、 R , 初始 $L = R = 1$, $sum = a[L]$
 - 当 $sum \geq S$ 时, 符合要求, 用 $(R-L+1)$ 更新答案, 且 $L++$; 若 $L = R$, 则 $L++, R++$
 - 当 $sum < S$ 时, 不符合要求, $R++$



尺取法

- $[5\ 1\ 3\ 5\ 10\ 7\ 4\ 9\ 2\ 8]$, $S = 15$



简要证明

- $[5\ 1\ 3\ 5\ 10\ 7\ 4\ 9\ 2\ 8]$, $S = 15$
- 根据算法流程可知
 - 左端点 L 能够遍历到区间中的所有点
 - 对于每个左端点 L 所形成的最终区间 $[L, R]$
 - $\text{sum}[L \sim R]$ 如果小于 S , 则令 R 右移
 - $\text{sum}[L \sim R]$ 如果大于等于 S , 此时 $\text{sum}[L \sim R] \geq S$, $\text{sum}[L \sim (R-1)] < S$
- 对于一个左端点 L , 最后找到了一个区间 $[L, R]$, R 是满足条件的最小值
- 因此答案区间一定会在上述尺取流程中出现

尺取算法总结

- 维护双指针（下标）
 - 尺取法，又称双指针法，即在遍历数组过程中，用两个相同方向进行扫描
- 什么情况下能使用尺取法？
 - 所求解答案为一个连续区间
 - 对于一个左端点 L ， $[L, n]$ 的区间是否满足条件是单调的
- 经典例题
 - 长度最小的区间和 $\geq S$ 的连续区间



5

例 2 讲 解

E x a m p l e 3

例2讲解

- 题意
 - 一个长度为 n 的字符串 s ，其中仅包含 'A', 'B', 'C', 'D' 四种字符 (n 为 4 的倍数)
 - 如果四种字符在字符串中出现次数均为 $n/4$ ，则其为一个平衡字符串
 - 现可以将 s 中连续的一段替换成任意字符，使其变为一个平衡字符串，问替换子串的最小长度？
- 样例
 - $s = \text{"AABC"}$
 - 转变为 "DABC" ，答案为 1

解题思路

- 是否可以用尺取法？
 - 所求解答案为一个连续区间 ✓
 - 对于一个左端点 L , $[L, n]$ 的区间是否满足条件单调 ✓
 - 当前 $[L, R]$ 满足要求, 则 $L++$
 - 当前 $[L, R]$ 不满足要求, 则 $R++$
- 因此可以用尺取法, 思考如何判断当前 $[L, R]$ 是否满足要求

解题思路

- 给定 $[L, R]$ ，判断是否满足要求？
 - 用 $\text{cntA}, \text{cntB}, \text{cntC}, \text{cntD}$ 分别记录在区间 $[L, R]$ 字符 'A', 'B', 'C', 'D' 的个数
 - 用 $\text{sumA}, \text{sumB}, \text{sumC}, \text{sumD}$ 表示字符 'A', 'B', 'C', 'D' 的总个数
 - 如果 $\text{sumA} - \text{cntA} > n/4$ 或者 $\text{sumB} - \text{cntB} > n/4$ 或者 $\text{sumC} - \text{cntC} > n/4$ 或者 $\text{sumD} - \text{cntD} > n/4$ ，则不符合条件



6

单调栈

Monotone Stack

单调栈

- 单调栈 = 单调 + 栈
- 单调
 - 单调递增 1 2 3 4 5 6 7...
 - 单调递减 7 6 5 4 3 2 1...
 - 单调非减 1 1 2 3 4 4 5...
 - 单调非增 7 7 6 5 4 4 3...
- 栈
 - 一种线性数据结构，支持 push 和 pop 操作
 - 已经在「数据结构与算法」课程中学习过
 - 满足先进后出 FILO 原则
- 单调栈 = 栈内元素 **自栈顶到栈底** 满足 **单调性**

单调栈

- 模拟单调递增栈的操作
 - 现有一数组 $[10, 3, 7, 4, 12]$ ，从左到右依次入栈。
 - 如果栈为 **空** 或者栈顶元素 **大于** 入栈元素，则入栈。
 - 否则，入栈则会破坏栈内元素的单调性，则需要将不满足条件的栈顶元素全部弹出后，将入栈元素入栈。
- 单调递减栈与之相反

单调栈

- 模拟单调递增栈的操作 []
 - 10 入栈时, 栈空, 入栈

单调栈

- 模拟单调递增栈的操作 [10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈

单调栈

- 模拟单调递增栈的操作 [3 10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈;

单调栈

- 模拟单调递增栈的操作 [10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈

单调栈

- 模拟单调递增栈的操作 [7 10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈
 - 4 入栈时, 栈顶元素 7 大于 4, 入栈

单调栈

- 模拟单调递增栈的操作 [4 7 10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈
 - 4 入栈时, 栈顶元素 7 大于 4, 入栈
 - 12 入栈时, 栈顶元素 4 小于 12, 弹栈;

单调栈

- 模拟单调递增栈的操作 [7 10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈
 - 4 入栈时, 栈顶元素 7 大于 4, 入栈
 - 12 入栈时, 栈顶元素 4 小于 12, 弹栈; 栈顶元素 7 小于 12, 弹栈;

单调栈

- 模拟单调递增栈的操作 [10]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈
 - 4 入栈时, 栈顶元素 7 大于 4, 入栈
 - 12 入栈时, 栈顶元素 4 小于 12, 弹栈; 栈顶元素 7 小于 12, 弹栈; 栈顶元素 10 小于 12, 弹栈;

单调栈

- 模拟单调递增栈的操作 []
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈
 - 4 入栈时, 栈顶元素 7 大于 4, 入栈
 - 12 入栈时, 栈顶元素 4 小于 12, 弹栈; 栈顶元素 7 小于 12, 弹栈; 栈顶元素 10 小于 12, 弹栈; 栈空, 入栈

单调栈

- 模拟单调递增栈的操作 [12]
 - 10 入栈时, 栈空, 入栈
 - 3 入栈时, 栈顶元素 10 大于 3, 入栈
 - 7入栈时, 栈顶元素 3 小于 7, 弹栈; 栈顶元素 10 大于 7, 入栈
 - 4 入栈时, 栈顶元素 7 大于 4, 入栈
 - 12 入栈时, 栈顶元素 4 小于 12, 弹栈; 栈顶元素 7 小于 12, 弹栈; 栈顶元素 10 小于 12, 弹栈; 栈空, 入栈

单调栈

- 单调递增栈的伪代码

```
1  INITIALIZE stack
2  FOR each element u DO
3      WHILE stack.size() > 0 and stack.top() <= u DO
4          stack.pop()
5      END
6      stack.push(u)
7  END
```

单调栈

- 单调栈的作用
 - 线性的时间复杂度
 - **单调递增栈** 可以找到往左/往右第一个比当前元素 **小** 的元素
 - **单调递减栈** 可以找到往左/往右第一个比当前元素 **大** 的元素
 - 可以求得以当前元素为最值的最大区间

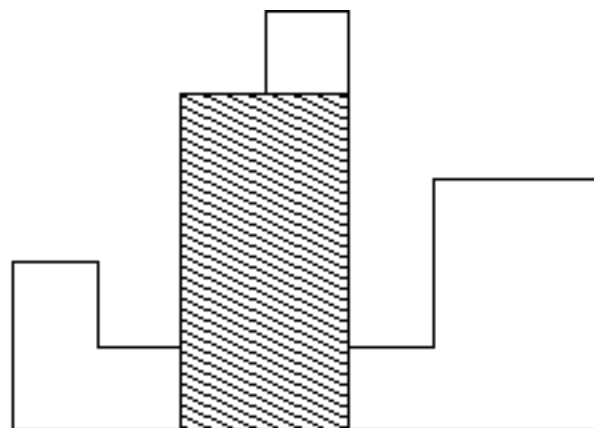
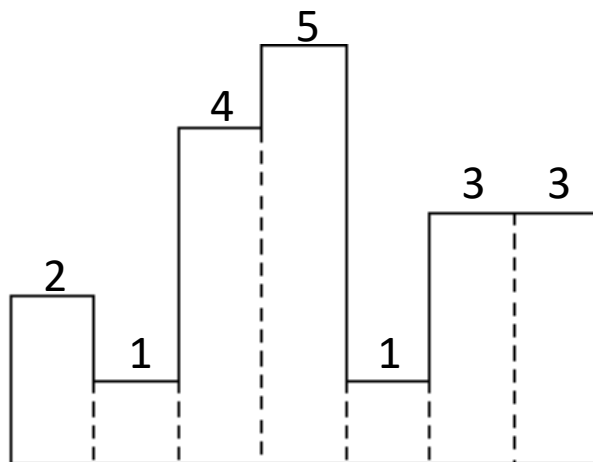


例 3 讲 解

E x a m p l e 1

例3讲解

- 给一个直方图，求直方图中的最大矩形的面积
- $1 \leq n \leq 100000$
- $n=7, a = [2 \ 1 \ 4 \ 5 \ 1 \ 3 \ 3]$
- $\text{Ans} = 8$



例3讲解

- 如果确定了一个矩形的左端点为 l ，右端点为 r ，那么矩形的高怎么确定？
 - 根据题意，高度不能超过 l 到 r 中的最小值
 - 要是整个矩形最大，那么高度就确定为 l 到 r 中的最小值
- 同理，如果确定了矩形的高度，那么左端点一定是越靠左，右端点越靠右，这个矩形的面积才可能最大
 - 左端点可以确定为往左数第一个小于此高度的点
 - 右端点可以确定为往右数第一个小于此高度的点
- 两遍单调栈处理出以每个点为高时的左右端点



单调队列

Monotone Queue

单调队列

- 单调队列 = 单调 + 队列
- 单调
 - 单调递增 1 2 3 4 5 6 7...
 - 单调递减 7 6 5 4 3 2 1...
 - 单调非减 1 1 2 3 4 4 5...
 - 单调非增 7 7 6 5 4 4 3...
- 队列
 - 一种线性数据结构，支持 push 和 pop 操作
 - 已经在「数据结构与算法」课程中学习过
 - 满足先进先出 FIFO 原则
- 单调队列 = 队列里的元素满足 **出队** 顺序的 **单调性**

单调队列

- 模拟单调递增队列的操作
 - 现有一数组 $[10, 3, 7, 4, 12]$ ，从左到右依次入队。
 - 如果队列为 **空** 或者队尾元素 **小于** 入队元素，则入队。
 - 否则，入队则会破坏队内元素的单调性，则需要将不满足条件的队尾元素全部出队后，将入队元素入队。
- 单调递减队列与之相反

单调队列

- 单调队列的维护过程与单调栈相似
- 区别在于
 - 单调栈只维护一端(栈顶), 而单调队列可以维护两端(队首和队尾)
 - 单调栈通常维护 **全局** 的单调性, 而单调队列通常维护 **局部** 的单调性
 - 单调栈大小没有上限, 而单调队列通常有大小限制
- 由于单调队列 **可以队首出队** 以及 **前面的元素一定比后面的元素先入队** 的性质, 使得它可以维护局部的单调性
- 当队首元素不在区间之内则可以出队
- 时间复杂度与单调栈一致

单调队列

- 单调递增队列的伪代码

```
1  INITIALIZE queue
2  FOR each element u DO
3      WHILE queue.size() > 0 AND queue.front() does not belong to the interval DO
4          queue.pop_front()
5      END
6      WHILE queue.size() > 0 AND queue.back() > u DO
7          queue.pop_back()
8      END
9      queue.push(u)
10 END
```



9

例 2 讲 解

E x a m p l e 2

例4讲解

- LZH 有一个长度为 n 的数列和一个大小为 k 的窗口, 窗口可以在数列上来回移动. 现在 LZH 想知道在窗口从左往右滑的时候, 每次窗口内数的最大值和最小值分别是多少.
- $1 \leq n \leq 1000000, 1 \leq k \leq n$

例4讲解

- 例如当数列为 [1 3 -1 -3 5 3 6 7], $k=3$ 时

Window position	Minimum value	Maximum value
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

例4讲解

- 如果查找全局的最小值, 可以使用单调栈, 尽管有些多余
- 现在要求查找窗口内的最小值, 是一个 **局部** 的概念
- 维护一个单调递增队列, 队列中的元素均属于当前窗口
- 当元素不属于当前窗口时, 将队首元素弹出即可
- 可以手动模拟一下整个过程

单调队列

- 模拟单调递增队列的操作 []
 - 1 入队时, 队空, 入队

单调队列

- 模拟单调递增队列的操作 [1]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队

单调队列

- 模拟单调递增队列的操作 [1 3]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队

单调队列

- 模拟单调递增队列的操作 [1]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队

单调队列

- 模拟单调递增队列的操作 []
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1

单调队列

- 模拟单调递增队列的操作 [-1]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队

单调队列

- 模拟单调递增队列的操作 []
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队

单调队列

- 模拟单调递增队列的操作 [-3]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值-3

单调队列

- 模拟单调递增队列的操作 [-3 5]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值-3
 - 3 入队时, 队尾元素 5 大于 3, 5出队

单调队列

- 模拟单调递增队列的操作 [-3]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值-3
 - 3 入队时, 队尾元素5大于3, 5 出队, 队尾元素 -3 小于 3, 3入队, 并记录当前最小值 -3

单调队列

- 模拟单调递增队列的操作 $[-3 \ 3]$
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值 -3
 - 3 入队时, 队尾元素 5 大于 3, 5 出队, 队尾元素 -3 小于 3, 3 入队, 并记录当前最小值 -3
 - 6 入队时, 队首元素 -3 的下标小于等于 $n-k=4$, -3 出队,

单调队列

- 模拟单调递增队列的操作 [3]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值-3
 - 3 入队时, 队尾元素5大于3, 5 出队, 队尾元素 -3 小于 3, 3入队, 并记录当前最小值 -3
 - 6入队时, 队首元素 -3 的下标小于等于 $i-k=4$, -3出队, 队尾元素 3小于6, 入队, 并记录当前最小值3

单调队列

- 模拟单调递增队列的操作 [3 6]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值-3
 - 3 入队时, 队尾元素5大于3, 5 出队, 队尾元素 -3 小于 3, 3入队, 并记录当前最小值 -3
 - 6入队时, 队首元素 -3 的下标小于等于 $n-k=4$, -3出队, 队尾元素3小于6, 入队, 并记录当前最小值3
 - 7入队时, 队尾元素6小于7, 入队, 并记录当前最小值3

单调队列

- 模拟单调递增队列的操作 [3 6 7]
 - 1 入队时, 队空, 入队
 - 3 入队时, 队尾元素 1 小于 3, 入队
 - -1 入队时, 队尾元素 3 大于 -1, 3 出队, 队尾元素 1 大于 -1, 1 出队, 队空, -1 入队, 并记录当前最小值 -1
 - -3 入队时, 队尾元素 -1 大于 -3, -1 出队, 队空, 入队, 并记录当前最小值 -3
 - 5 入队时, 队尾元素 -3 小于 5, 入队, 并记录当前最小值-3
 - 3 入队时, 队尾元素5大于3, 5 出队, 队尾元素 -3 小于 3, 3入队, 并记录当前最小值 -3
 - 6入队时, 队首元素 -3 的下标小于等于 $n-k=4$, -3出队, 队尾元素3小于6, 入队, 并记录当前最小值3
 - 7入队时, 队尾元素6小于7, 入队, 并记录当前最小值3



为天下储人才
为国家图富强

感谢收听

Thank You For Your Listening