队列与单调队列

先进先出 FIFO、循环队列取模与 $O(n)$ 滑动窗口极值

HGNU-ACM 实验室算法组

配套：循环队列防假溢出 $+$ 双端队列 `deque` $+$ 单调队列
$O(n)$ 论证 $+$ 4 道真题精讲





本章目录（1/2）

普通队列 FIFO、循环队列与双端队列

单调队列：$O(n)$ 解决定长滑动窗口极值

承接第 1 章《栈与单调栈》，前两节讲透普通队列 FIFO、循环队列取模防假溢出、双端队列 deque，以及均摊 $O(n)$ 的单调队列滑动窗口极值。

本章目录（2/2）

四大队列与单调队列真题精讲

常见错误、检查清单与练习安排

后两节逐题精讲《周末舞会》《围圈报数》《奇怪的电梯》《最敏捷的机器人》，并附上高频防错对照与第 3 章《树、二叉树与堆》预告。

承接第 1 章：栈管“回头”，队列管“公平排队”

- 在第 1 章《栈与单调栈》中，栈像一摞洗好的盘子——最后放进去的最先拿出来（**LIFO**），专治括号嵌套、后缀求值等就近回溯问题。
- 可生活和竞赛中还有另一类常见秩序：谁先排进来，就必须谁先办完出去！
- 比如食堂打饭排队、医院叫号、或是 BFS 里先处理第 1 层的点再处理第 2 层的点——这些全部都要使用队列（**Queue**）。

小对比：依次放入 $10, 20, 30$ 再全部取出

栈取出来的顺序是 $30 \to 20 \to 10$；而队列取出来的顺序一定是：$10 \to 20 \to 30$！

本页要点

栈是“一端进出”（后来先出），队列是“队尾进、队头出”（先来先出 FIFO）！

生活引入：食堂窗口排队与银行叫号

场景一：食堂打饭窗口

- 同学 A 最早排在最前面（队头），B 和 C 依次排在后面。
- 阿姨永远先给站在队头的 A 打饭，打完 A 就离开队伍。
- 新来的同学 D 只能老老实实排到最后面（队尾），不能插队。

场景二：医院排队叫号

- 病人取号后按编号排成一列，叫号机每次只喊走最靠前的一位。
- 就算队伍前面走掉了 10 个人，后面新来的人依然排在当前队列末尾。
- “到达的先后顺序”在队伍里被原封不动地保留下来。

本页要点

队列只允许在队头查看或删除，在队尾追加新人，绝对不允许从中间插队或抽人！

暴力痛点：为什么不能用 vector 每次删第 0 格模拟排队？

有些同学想：我直接用数组或 vector<int> q，入队写 q.push_back(x)，出队写 q.erase(q.begin()) 不就行了吗？

- 看清背后的代价：当你用 erase(q.begin()) 删掉下标 0 的元素时，后面的所有 m 个人都必须全体往前挪动一格，单次就要 $O(n)$ 步！
- 假设有 10^5 个人先全部排进队列，然后再依次叫号出队 10^5 次：

$$\text{总搬移次数} \approx (10^5 - 1) + (10^5 - 2) + \cdots + 1 \approx 5 \times 10^9 \text{ 步！}$$

- 这远远超过了评测机 1 秒约 10^8 次运算的极限，直接 **TLE**！
- 正确解法：不搬移后面的人！只用一个下标 head 记录“队头挪到哪了”，出队只需 head++，每次只要 $O(1)$（**1** 步）！

本页要点

用移动 head 指针代替全体左移，队列的入队和出队就能从 $O(n)$ 降到严格的 $O(1)$！

新术语大白话：队头、队尾、入队、出队、FIFO

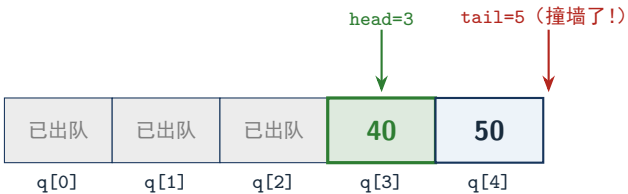
- 队列（**Queue**）：一头只进、另一头只出的排队结构。
- 队头（**Front / Head**）：队伍最前面等待办理的那个位置；也是唯一能读出或删除元素的一端。
- 队尾（**Back / Tail**）：队伍最后面，新元素只能从这一端排进来。
- 入队（**Push**）：把新元素放到队尾后面（排队）。
- 出队（**Pop**）：把当前队头元素移出队伍（办完离开）。
- **FIFO**（**First In, First Out**）：“先进先出”的英文缩写，先入队的一定先出队。

本页要点

队列的四个核心动作与栈对称：判空 `empty`、看头 `front`、入队 `push`、出队 `pop`！

TikZ 图解：普通数组队列的“假溢出”危机

如果一直向右移动 head 和 tail，数组明明前面还空着 3 个位置，却会撞到右边界！



前面 0,1,2 全空着，tail 却无法再右移——这就是假溢出！

本页要点

直线数组一旦前面出队腾出空位，后面的 tail 却顶到天花板了，白白浪费前面空间！

循环队列原理：把直跑道弯成环形操场（tail + 1）% M

怎么把浪费掉的 q[0]，q[1]，q[2] 重新利用起来？只需让指针走到数组末尾 $M-1$ 后自动绕回 0！

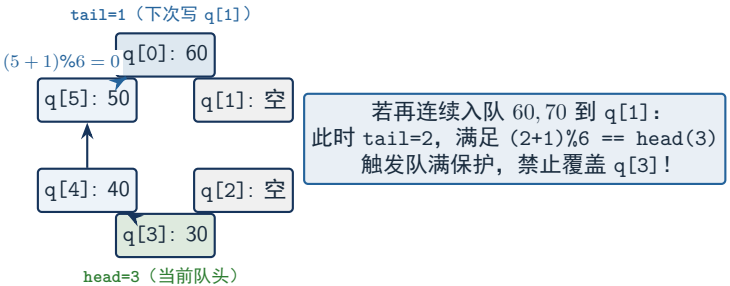
- 前进统一写成取模：
  - 入队后尾指针前进一格：`tail = (tail + 1) % M;`
  - 出队后头指针前进一格：`head = (head + 1) % M;`
- 怎么区分“队空”和“队满”？
  - 如果让 M 个格子全部塞满，满的时候也会出现 `head == tail`，和队空一模一样！
  - 经典约定（牺牲 1 格空位）：数组开长度 M，最多只装 $M-1$ 个元素！
  - 于是：队空条件是 `head == tail`；队满条件是 `(tail + 1) % M == head`！

本页要点

记住两个铁律：取模让 $M-1$ 的下一格回到 0；故意留 1 个空槽，就能一眼分清队空和队满！

TikZ 图解：循环队列 $M = 6$ 环形槽位与绕回箭头

开大小为 $M = 6$ 的数组 q[0..5]（最多存 5 个元素），看 tail 如何绕回 0 以及何时判满：



本页要点

当 tail 追到 head 的紧前面一格（即 (tail + 1) % M == head）时立刻停手，绝不踩进 head！

手算跟踪表：循环队列 $M = 5$ 的 6 步操作（含绕回与判空）

设数组长度 $M = 5$（最多装 4 个元素），初始 `head = 0, tail = 0`：

步序	操作	判满/判空检查	操作后 (head, tail)	队列有效元素（头 → 尾）
1	push(1), push(2), push(3)	$(2+1)\%5 \neq 0$	(0, 3)	[1, 2, 3]（占 0,1,2）
2	pop(), pop()	head != tail	(2, 3)	[3]（0,1 空出）
3	push(4), push(5)	$(4+1)\%5 \neq 2$	(2, 0)	[3, 4, 5]（tail 从 4 绕回 0!）
4	push(6)	$(0+1)\%5 = 1 \neq 2$	(2, 1)	[3, 4, 5, 6]（此时占 4 格）
5	尝试 push(7)	(1+1)%5 == 2 为真	(2, 1)	队满拦截！拒绝写入
6	出队 4 次后判空	head == tail（1 == 1）	(1, 1)	队列空，循环终止（防 RE!）

本页要点

第 3 步 push(5) 写进 q[4] 后 tail 绕回 0，成功复用了第 2 步腾出的 q[0] 槽位！

为什么对：循环队列的不变量三段论证

阶段	保证成立的事实（不变量）
初始化	`head = 0`，`tail = 0`，此时 `head == tail`，队列恰好有 0 个有效元素。
保持	每次入队仅在 `(tail+1)%M != head` 时写入 `q[tail]` 并令 `tail = (tail+1)%M`；每次出队仅在 `head != tail` 时令 `head = (head+1)%M`。则有效元素始终沿顺时针方向依次存放在 `head` 到 `tail` 前一格，个数为 `(tail - head + M) % M`。
终止	当 `head == tail` 时元素个数为 0，停止读取或出队，彻底避免空队越界！

本页要点

顺时针环上元素总数公式是 `(tail - head + M) % M`（加 M 是为了防止 `tail < head` 时减成负数）！

模板代码（1/2）：手写循环队列防假溢出模板

```
const int M = 100005;                    // 最多存 M-1 个元素
int q[M], head = 0, tail = 0;            // head==tail 代表队空

bool empty() { return head == tail; }
bool full()  { return (tail + 1) % M == head; }

bool push(int x) {
    if (full()) return false;            // 队满拒绝入队
    q[tail] = x;                         // 写入当前队尾槽位
    tail = (tail + 1) % M;               // 尾指针顺时针绕回
    return true;
}
void pop() { if (!empty()) head = (head + 1) % M; } // 队非空才移动头指针
```

本页要点

读取队头前务必先判 `!empty()` 再读 `q[head]`；多组测试记得每组重置 `head = tail = 0`！

模板代码（2/2）：STL queue<int> 安全模板

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    queue<int> q;                       // 底层自动管理内存，不怕假溢出
    q.push(10); q.push(20);             // 依次排到队尾：10 在前，20 在后
    while (!q.empty()) {                // 只要队伍没空就继续叫号
        int x = q.front();              // 读取当前队头
        q.pop();                        // 弹出队头（pop 没有返回值！）
        cout << x << '\n';              // 输出顺序必为 10，然后 20
    }
    return 0;
}
```

本页要点

日常做题直接用 queue<int> 最省心；只有在手写环形缓冲区或题目要求固定容量环形排队时才写取模版！

双端队列 deque：左右两扇门都能进出的通道

普通 queue 只准队尾进、队头出；而接下来学单调队列时，我们既要从队头踢走过期元素，又要从队尾踢走更劣元素，这就需要双端队列 deque！



本页要点

deque<int> dq 同时拥有头尾两侧的 push/pop，而且全部只需 $O(1)$ 时间，还支持下标 dq[i]！

选型对照：手写循环队列 vs STL queue vs deque

能力需求	手写循环队列	STL queue	STL deque
队尾入队	`q[tail]=x; tail=(tail+1)%M`	`q.push(x)`	`dq.push_back(x)`
队头出队	`head=(head+1)%M`	`q.pop()`	`dq.pop_front()`
队尾删/队头插	需改指针减法取模	不支持	`pop_back()` / `push_front()`
查看队头/队尾	`q[head]` / `q[(tail-1+M)%M]`	`q.front()` / `q.back()`	`dq.front()` / `dq.back()`
适用场景	极限卡常、固定环长	周末舞会、标准 BFS	单调队列、0-1 BFS

本页要点

遇到“只进尾出头”就用 queue；一旦需要“两头都能进或出”，立刻换 deque！

deque 常用接口与手算小例

```
deque<int> dq;                                  // 双端队列，支持首尾 O(1) 增删
dq.push_back(20);                               // 队尾加入 20: [20]
dq.push_front(10);                              // 队头加入 10: [10, 20]
dq.push_back(30);                               // 队尾加入 30: [10, 20, 30]
cout << dq.front() << ' ' << dq.back() << '\n'; // 输出 10 30
dq.pop_front();                                 // 弹出队头 10: [20, 30]
dq.pop_back();                                  // 弹出队尾 30: [20]
```

- 判空用 dq.empty()，元素个数用 dq.size()；与 queue 一样，pop_front() 和 pop_back() 都没有返回值。

本页要点

单调队列就是靠 dq.pop_front() 踢过期队头、靠 dq.pop_back() 踢弱队尾！

deque 易错提醒：为什么不直接拿它当万能数组用？

- 易错 1：空 deque 直接读头尾或弹出
  - 错误：dq.front() 或 dq.pop_back() 前未检查 !dq.empty() $\Longrightarrow$ 直接触发 **RE**（未定义行为）。
- 易错 2：把“队头”和“队尾”方向搞反
  - 在单调队列里，约定新元素永远从 back（队尾）进，老元素过期从 front（队头）出；如果一会 push_front 一会 pop_back，下标单调性会彻底乱套。
- 易错 3：盲目用 dq[i] 随机访问代替 vector
  - deque 底层是分段连续块，随机访问常数比 vector 慢；不需要“队头删除”时优先用 vector。

本页要点

用 deque 前先定死方向：哪一端是“最老的队头”，哪一端是“最新的队尾”，全程不改！

上机验证：3 组测试与反事实追问

测试手写循环队列模板（设数组容量 $M = 4$，即最多同时容纳 3 个元素）：

1. 正常先进先出：依次 `push(10)`，`push(20)`，`pop()`，`front()` $\Longrightarrow$ 应输出 20。
2. 跨越末尾回绕：在 $M = 4$ 中先 `push(1)`，`push(2)`，`push(3)`，再 `pop()` 两次，再 `push(4)`，`push(5)` $\Longrightarrow$ 队内依次为 3，4，5，无越界！
3. 空队列保护：连续弹出全部元素后，`empty()` 为 `true`，此时再调用 `pop()` 被安全拦截。

反事实追问

如果循环队列不留 1 个空位，允许把 M 个槽位全部塞满，只靠 `head == tail` 会发生什么？

答：队空时 `head == tail`，塞满 M 个元素绕一圈回来后 `tail` 再次追上 `head`，也是 `head == tail`，程序根本分不清队列到底是空的还是满的！

易错对照：普通队列与循环队列四大错误

错误类型	常见错误写法	评测结果与正确写法
空队列取头或弹头	不写 `!q.empty()` 直接 `q.front()`	立刻越界或段错误 **RE**；先判 `!q.empty()`
用 vector 删首项	循环里写 `v.erase(v.begin())`	10^5 数据直接超时 **TLE**；改用 queue 或 head++
循环队列漏取模	只写 `tail++` 忘了 `% M`	出队再多依然下标越界 **RE**；必须 `(tail+1)%M`
直线数组开小了	最多入队 10^5 次开 `q[1005]`	哪怕队列最多同时 10 人，总入队次数多仍 **RE**！改循环队列或开够总入队次数

本页要点

使用不取模的普通数组队列时，数组大小必须按历史累计入队总次数开，而不是按当前最大人数开！

小例子引入：宽 $k=3$ 的窗口一路向右滑，每步最大是谁？

手算体验：序列 $a=[1,3,-1,-3,5,3]$（下标 $0\sim5$），窗口长度 $k=3$

- 窗口 $[0..2]=[1,3,-1]\implies$ 最大值是 3。
- 窗口 $[1..3]=[3,-1,-3]\implies$ 最大值是 3。
- 窗口 $[2..4]=[-1,-3,5]\implies$ 最大值是 5（注意：刚才的 3 已经滑出窗口左边了！）。
- 窗口 $[3..5]=[-3,5,3]\implies$ 最大值是 5。

- 窗口一共滑出 $n-k+1=4$ 个位置，输出序列为 3，3，5，5。
- 这只有 6 个数，如果数组长 $n=10^5$、窗口宽 $k=5\times10^4$，怎么快速算出全部窗口的最大值？

本页要点

窗口每右移一格，左边丢掉 1 个旧数、右边进来 1 个新数；我们要找一种能一趟搞定全部窗口的方法！

暴力痛点：每轮重扫窗口为什么会 TLE？（兼对比堆）

- 暴力双层循环：外层枚举窗口右端点 $i = k-1 \dots n-1$，内层从 $i-k+1$ 扫到 i 求最大值。
  - 当 $n = 10^5, k = 5 \times 10^4$ 时，窗口数 $n-k+1 \approx 5 \times 10^4$，每次扫 5×10^4 个数。
  - 总比较次数 $\approx 2.5 \times 10^9$ 次，远超 1 秒 10^8 次运算极限，必定 **TLE**！
- 用平衡树 / 优先队列（堆）：虽然能降到 $O(n \log n)$，但在 $n = 10^6$ 时 $\log n \approx 20$，常数偏大仍可能卡常。
- 单调队列：直接把复杂度压到极致的 $O(n)$，平均每个数只处理常数步，再大的 $n = 10^6$ 也能毫秒级 AC！

本页要点

定长滑动窗口最值问题的理论最优解就是 $O(n)$ 单调队列，比堆少一个 $\log n$！

新术语大白话：滑动窗口、队头过期、队尾淘汰

- 滑动窗口：长度为 k 的连续区间 $[i-k+1, i]$，右端点 i 每加 1，整个框框就向右平移 1 格。
- 队头过期：当新位置是 i 时，所有下标 $\le i-k$ 的元素都已经跌出了窗口左边界，必须从队头踢出去。
- 队尾淘汰：新元素 i 准备从队尾进队时，如果发现队尾元素 j 能力不如它（$a[j] \le a[i]$），就把老兵 j 从队尾踢掉。
- 单调队列：用双端队列 `deque` 维护一条从队头到队尾数值严格递减的候选下标队伍，队头永远是当前窗口最大值的下标！

本页要点

记住这两条门规：队头查“年龄超期没”（$\le i-k$），队尾查“能力输给新人没”（$\le a[i]$）！

核心洞察：为什么“又老又弱”的人可以立刻踢出队伍？

部队选“当前窗口最强新人王”的比喻

把下标 i 看作入伍年份（越大越年轻），把 $a[i]$ 看作战斗力。求窗口 $[i-k+1, i]$ 最大值时，队尾老兵 j 和新兵 i（$j < i$）对比：

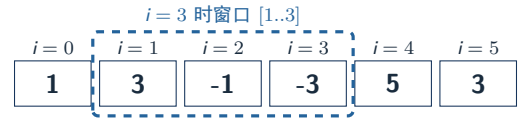
- 老兵 j 比新兵 i 入伍更早，意味着老兵 j 一定比 i 更早退役（滑出窗口左边界）。
- 如果这个老兵 j 的战斗力还不如（或等于）新兵 i（即 $a[j] \le a[i]$），那只要有 i 在场，未来包含 j 的每一个窗口也必定包含 i！
- 所以老兵 j 这辈子再也没有机会当老大，直接从队尾请他回家！

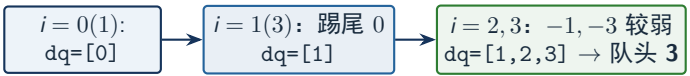
本页要点

留下来的人必须是“越年轻的越弱，老的比小的更强”——只有老家伙强过新兵，才有资格等新兵上位前暂居榜首！

TikZ 图解（1/2）：前 4 步（$i = 0 \ldots 3$）队尾淘汰过程

序列 $a = [1, 3, -1, -3, 5, 3]$，$k = 3$。观察 $i = 1$（值 3）到来时怎么把 0（值 1）从队尾踢掉：



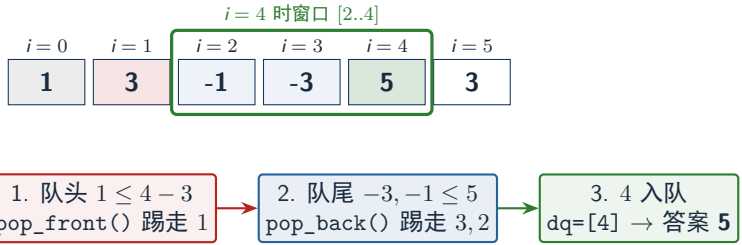


本页要点

$i = 2(-1)$ 和 $i = 3(-3)$ 虽然打不过队头 3，但它们比 3 年轻！等 3 退休后它们还有机会，所以暂时留队尾！

TikZ 图解（2/2）：$i = 4$（值 5）时的“头踢过期 + 尾踢弱者”

窗口滑到 $[2..4]$（下标 4 的新兵 5 到达），左边界是 $4 - 3 + 1 = 2$，因此下标 ≤ 1 全过期：



本页要点

这一步把单调队列的精髓演全了：前面因为“超龄”踢掉 1，后面因为“打不过 5”踢掉 3 和 2！

手算跟踪表：$a=[1,3,-1,-3,5,3]$，$k=3$（含未满与终止）

dq 内存储下标（括号内写对应数值），时刻保持从队头到队尾数值严格递减：

i ($a[i]$)	过期 front<=i-3	踢弱尾 <=a[i]	入队后 dq	满 3 项才输出
0 (1)	队空	队空	$[0_{(1)}]$	窗口未满
1 (3)	假（$0 \le -2$ 假）	$a[0]=1\le 3$，踢 0	$[1_{(3)}]$	窗口未满
2 (-1)	假（$1 \le -1$ 假）	无	$[1_{(3)}, 2_{(-1)}]$	[0..2] 最大 **3**
3 (-3)	假（$1 \le 0$ 假）	无	$[1_{(3)}, 2_{(-1)}, 3_{(-3)}]$	[1..3] 最大 **3**
4 (5)	真（$1 \le 1$ 踢 1）	踢 $3_{(-3)}$ 再踢 $2_{(-1)}$	$[4_{(5)}]$	[2..4] 最大 **5**
5 (3)	假（$4 \le 2$ 假）	无	$[4_{(5)}, 5_{(3)}]$	[3..5] 最大 **5**
结束	$i=6$ 退出循环	无	无需清空	全部答案 **3, 3, 5, 5**

本页要点

前 $k-1$ 步（即 $i<k-1$）只维护队列不输出；从 $i+1\ge k$ 开始每轮队头 a[dq.front()] 就是答案！

均摊 $O(n)$ 论证：两个 while 嵌套为什么依然是 $O(n)$？

主循环是 `for (int i = 0; i < n; ++i)`，里面还有两个 `while`，时间复杂度怎么证明？

- 把每个下标 $i \in \{0, 1, \ldots, n-1\}$ 看成进站旅客：
  1. 在第 i 轮只有一行 `dq.push_back(i);`，因此每个下标一辈子入队恰好 1 次，共 n 次。
  2. 无论是从队头 `pop_front()` 被踢走，还是从队尾 `pop_back()` 被踢走，一旦出去就再也不可能回来！所以每个下标最多出队 1 次，两处 `while` 循环体合起来最多执行 n 次！
- 总入出队操作严格不超过 $2n$ 次，时间复杂度是绝对稳定的 $O(n)$！

本页要点

不管中间某一步 `pop_back` 连续踢掉多少人，踢掉的都是以前攒着没出队的人，平均每人总账不超过 2 步！

循环不变量：为什么队头一定是当前窗口最大值的下标？

阶段	保证成立的事实（不变量）
初始化	开始前 dq 为空，窗口内尚无元素，不变量平庸成立。
保持	每轮处理完 i 后：(1) dq 中下标都在当前合法窗口 $[\max(0, i-k+1), i]$ 内且严格递增；(2) 对应数值从队头到队尾严格递减；(3) 被踢掉的弱元素在窗口中都有更强且更晚过期的替代者留在队里，因此当前窗口真正最大值一定还在 dq 中，且它必是单调递减队列的第一个元素 dq.front()！
终止	$i = n$ 退出循环时，所有长度恰好为 k 的窗口 $[i-k+1, i]$ ($i \geq k-1$) 都已在对应轮次输出过正确最大值。

本页要点

“窗口内所有合法候选中，没被更强新兵淘汰的都在队里，而队里最强的只能蹲在队头”，这就是正确性保证！

关键追问：为什么 dq 里必须存“下标”而不是“数值”？

设想某同学偷懒，直接往 dq 里放数值：dq.push_back(a[i])，看看会立刻在哪一步翻车：

- 取原题样例 $a = [1, 3, -1, -3, 5, 3]$，$k = 3$ 稍作修改，变成 $a = [1, 3, -1, -3, 2, 3]$：
  - 在 $i = 4$（窗口 $[2..4] = [-1, -3, 2]$）时，下标 1 的那个 3 其实已经滑出窗口左边了！
  - 如果你的队列里只存着一个数字 3，没有下标 1，你怎么知道这个 3 到底是在窗口外的下标 1，还是在窗口内的下标 2？你完全无法判断它该不该从队头滚蛋！
- 存下标 idx 后，不仅能写出 dq.front() <= i - k 精准踢出过期的队头，还能用 a[idx] 随时读回原数值！

本页要点

没有下标，就不知道队头什么时候过期！单调队列里必须存下标，比较大小时才写 a[dq.back()]！

模板代码：单调队列求定长滑动窗口最大值

```
deque<int> dq;                                  // 队列里必须存下标!
for (int i = 0; i < n; ++i) {
    // 1. 先检查队头下标是否已经滑出窗口左边界 i - k + 1
    while (!dq.empty() && dq.front() <= i - k) dq.pop_front();
    // 2. 再从队尾踢掉所有不如当前新值 a[i] 的元素，保持严格递减
    while (!dq.empty() && a[dq.back()] <= a[i]) dq.pop_back();
    dq.push_back(i);                            // 3. 当前下标入队尾
    if (i + 1 >= k) {                           // 4. 窗口满 k 个才输出队头最大值
        cout << a[dq.front()] << '\n';
    }
}
```

本页要点

背下这四步顺序： 踢过期队头 $\rightarrow$ 踢弱队尾 $\rightarrow$ 当前下标入队 $\rightarrow$ 满 k 输出队头！

变体对比：求最小值怎么改？两步 while 顺序能换吗？

- 怎么改成求滑动窗口最小值？
  - 只需把第 2 步队尾淘汰的条件 `a[dq.back()] <= a[i]` 改成 `a[dq.back()] >= a[i]`！
  - 队列变成从队头到队尾严格递增：比新来的 $a[i]$ 还大（或相等）的老元素绝不可能当最小值，全从队尾踢掉，队头就是窗口最小值！
- 第 **1** 步（踢过期）和第 **2** 步（踢弱尾）能对调吗？
  - 对定长滑动窗口最大值/最小值而言，对调不影响结果，因为新元素 i 肯定在窗口内；但永远先踢过期再踢弱尾逻辑更干净，且不会把 i 误当成窗口内已存在元素，写固定顺序最不容易错！
  - 注意：步骤 **3**（入队）绝不能放到步骤 **2** 前面，否则你刚放进去的自己就把自己比错了；步骤 **4**（输出）必须在全部清理完之后！

本页要点

求最大值队尾踢 `<= a[i]`；求最小值队尾踢 `>= a[i]`；若同时要最大和最小，就开两个 deque 同步跑！

上机验证：3 组必测数据与反事实追问

用上面模板跑以下 3 组输入（格式：n, k 然后 n 个整数），核对全部输出：

1. 标准波动：$n = 6, k = 3,\ a = [1, 3, -1, -3, 5, 3] \implies$ 应输出 3，3，5，5。
2. 最小窗口边界：$n = 4, k = 1,\ a = [7, 2, 9, 4] \implies$ 每个窗口只有自己，应输出 7，2，9，4。
3. 严格递减（专测队头滑出）：$n = 5, k = 3,\ a = [9, 7, 5, 3, 1] \implies$ 应输出 9，7，5。

反事实追问

如果在上面第 3 组严格递减测试 $[9, 7, 5, 3, 1],\ k = 3$ 中，不小心把过期条件 `dq.front() <= i - k` 错写成 `dq.front() < i - k`，输出会变成什么？
答：$i = 3$ 时 $0 < 0$ 为假，本该滑出窗口的下标 0（值 9）赖在队头没走！输出会错误地变成 9，9，7（**WA**）！

易错对照与线性结构全景过渡

错误 / 场景	常见错误写法	正确做法与原因
把数值塞进队列	`dq.push_back(a[i])`	无法判断队头是否过期滑出，必须存下标 `i`
过期条件写成 <	`dq.front() < i - k`	窗口 $[i-k+1, i]$ 外最近失效下标是 `i - k`，必须写 `<=`
提前输出未满窗口	忘了 `if (i + 1 >= k)`	前 $k-1$ 步窗口不足 k 个数，绝不能提前打印
多处写法没判非空	漏写 `!dq.empty()`	访问空队头/尾直接段错误 **RE**

四件线性神器怎么选？（兼预告下一章）

遇到“最近嵌套/回溯”用栈；遇到“左/右侧首个更大或更小”用单调栈；遇到“先来先办/BFS”用队列；遇到“定长滑动窗口极值”用单调队列。下一章我们将走出线性队伍，进入树形分支结构《树、二叉树与堆》!

例 1 | 周末舞会（1332）[入门]：先读懂要求

大白话题意

舞会上，男士们和女士们进入舞厅时各自排成一队。每首舞曲开始时，依次从男队队头和女队队头各出 1 人配成舞伴；跳完这首后，两人各自排回本队队尾等待下一轮。给出男队人数、女队人数和舞曲数目，输出前 k 首舞曲的配对编号。

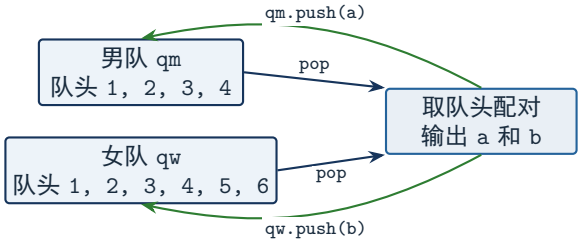
- 输入：第 1 行两个正整数 *men*, *women*（两队人数）；第 2 行正整数 *songs*（舞曲数目）。
- 处理：用两个普通队列模拟男队与女队，每轮各弹出队头并立刻放回队尾。
- 输出：共 *songs* 行，每行两个整数，表示本首舞曲的男、女编号。

本页要点

原题样例：输入 `4 6 / 7`，输出前 4 首 `1 1` → `4 4`，第 5 首男队绕回 `1 5`，第 7 首女队绕回 `3 1`。

例 1 | 怎么想到的？两支独立队列的“出队再入队”

如果用两个下标变量分别取模也可以做，但用两个普通队列最贴合“排队”直觉：



- 队伍初始化依次放入 $1 \ldots$ *men* 和 $1 \ldots$ *women*。
- 每一首舞曲：从 qm 弹出队头 *a* 再 qm.push(a)；从 qw 弹出队头 *b* 再 qw.push(b)，输出 a b。

本页要点

“队头出来办事、办完立刻排回队尾”是模拟环形轮转最稳的队列写法！

例 1 | 样例手推跟踪表：输入 4 6 / 7

初始 qm = [1, 2, 3, 4]，qw = [1, 2, 3, 4, 5, 6]，逐轮配对并排回队尾：

舞曲	配对输出	操作后 qm	操作后 qw	说明
第 1 首	1 1	[2, 3, 4, 1]	[2, 3, 4, 5, 6, 1]	双方队头 1 回到队尾
第 2 首	2 2	[3, 4, 1, 2]	[3, 4, 5, 6, 1, 2]	依次轮转
第 3 首	3 3	[4, 1, 2, 3]	[4, 5, 6, 1, 2, 3]	依次轮转
第 4 首	4 4	[1, 2, 3, 4]	[5, 6, 1, 2, 3, 4]	男队完成第一轮
第 5 首	1 5	[2, 3, 4, 1]	[6, 1, 2, 3, 4, 5]	男 1 第二次上场
第 6 首	2 6	[3, 4, 1, 2]	[1, 2, 3, 4, 5, 6]	女队完成第一轮
第 7 首	3 1	[4, 1, 2, 3]	[2, 3, 4, 5, 6, 1]	结束，共输出 7 行

本页要点

两队人数不同时互不干扰，各自按自己的长度周期循环！

例 1 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
读入 men，women，songs	两个队列 qm，qw	两队人数和舞曲数是否读对了行？
按 $1 \sim men$、$1 \sim women$ 入队	qm=[1..men], qw=[1..women]	编号是从 1 开始而不是从 0 开始吗？
循环 songs 次出队再入队	每轮输出 a b	先 front() 再 pop() 再 push() 吗？

相对基础队列模板改了哪里？

1. 普通模板只管“出队就走”，本题出队后必须立刻 push 回队尾，保证队列长度永远不变。
2. 需要同时维护两支彼此独立的队列 qm 和 qw。

例 1 | 完整可编译代码（1/2）：读入与两队初始化

以下源码与 code/weekend_dance.cpp 完全一致：

```cpp
#include <bits/stdc++.h>
using namespace std;
// 一本通 1332 周末舞会：循环队列模拟两支队伍队头配对后再排到队尾
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int men, women, songs;
    if (!(cin >> men >> women >> songs)) return 0;
    queue<int> qm, qw;
    for (int i = 1; i <= men; ++i) qm.push(i);
    for (int i = 1; i <= women; ++i) qw.push(i);
```

本页要点

两个 for 循环均从 1 到人数，确保编号与题目样例一致。

例 1 | 完整可编译代码（2/2）：循环配对与输出

接上一页 code/weekend_dance.cpp 结尾：

```
    for (int i = 0; i < songs; ++i) {
        int a = qm.front(); qm.pop(); qm.push(a);   // 出队配对后立刻回到队尾等待下
    次
        int b = qw.front(); qw.pop(); qw.push(b);
        cout << a << ' ' << b << '\n';
    }
    return 0;
}
```

本页要点

两队初始人数均 ≥ 1 且每次弹出后立刻塞回，因此循环过程中队列永远非空。

例 1 | 易错点与 3 组自测边界

- 易错 **1**：出队配对后忘了 `qm.push(a)`，导致几轮后队列变空引发 **RE**。
- 易错 **2**：编号从 0 开始入队，结果第 1 首输出 `0 0` 而不是 `1 1`，导致 **WA**。
- 易错 **3**：只用了一层 $O(1)$ 取模却忘了加 1，写出 `i % men`。

3 组必跑自测边界

1. 原题样例：`4 6 / 7` $\Longrightarrow$ 输出 7 行，首行 `1 1`、末行 `3 1`。
2. 单人队边界：`1 3 / 4` $\Longrightarrow$ 男生一直是 1，输出 `1 1`，`1 2`，`1 3`，`1 1`。
3. 等长边界：`2 2 / 3` $\Longrightarrow$ 输出 `1 1`，`2 2`，`1 1`。

例 2 | 围圈报数（1334）[入门]：先读懂要求

大白话题意

有 n 个人（$n \le 100$）编号 $1, 2, \ldots, n$ 依次围成一圈。从第 1 个人开始报数，数到第 m 个人就出列；然后从出列的下一个人重新从 1 开始报数，数到第 m 个人再出列……如此反复直到所有人都出列。按顺序输出出列人的编号。

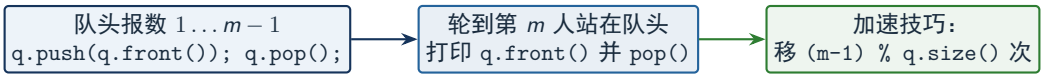
- 输入：两个正整数 n 和 m。
- 处理：用队列模拟圆圈，每轮把前 $m-1$ 个报数的人移到队尾，再弹出第 m 个队头。
- 输出：一行 n 个整数，空格隔开，表示出列顺序。

本页要点

原题样例：输入 `4 17`，输出 `1 3 4 2`。注意 $m = 17$ 远大于 $n = 4$！

例 2 | 怎么想到的？把圆圈剪开成队列 + 取模跳过整圈

如果用数组标记“是否出列”不断绕圈找人，代码分支多还容易写错下标。把圆圈从当前报数起点剪开拉直，就是一个普通队列：



- 关键优化：如果 $m = 10^9$ 而当前圈里只有 $q.size() = 4$ 个人，绕完整一圈等于位置完全没变！
- 因此每轮只需要把队头移到队尾 `step = (m - 1) % q.size()` 次，绝不空转绕圈！

本页要点

用 `(m - 1) % q.size()` 跳过多余整圈，即使 m 极大也能稳稳不超时！

例 2 | 样例手推跟踪表：输入 4 17

初始 `q = [1, 2, 3, 4]`，$m = 17$，每轮先算 `step = (17 - 1) % q.size()`：

轮次	轮初 q	step = 16 % size	移到队尾的人	出列（当前队头）
第 1 轮	[1, 2, 3, 4]	$16\%4 = 0$	无	**1**（剩 [2, 3, 4]）
第 2 轮	[2, 3, 4]	$16\%3 = 1$	2 移到队尾 → [3, 4, 2]	**3**（剩 [4, 2]）
第 3 轮	[4, 2]	$16\%2 = 0$	无	**4**（剩 [2]）
第 4 轮	[2]	$16\%1 = 0$	无	**2**（队列空，结束）

本页要点

4 轮依次输出 `1 3 4 2`，与原题样例输出完全一致！

例 2 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
将 $1 \sim n$ 依次入队	`q = [1, 2, ..., n]`	队列初值是否按从小到大入队？
计算本轮移动次数	`step = (m - 1) % q.size()`	为什么是 $m-1$ 而不是 m？（第 m 人不出尾要出列！）
弹出并控制空格格式	`first` 标记控制空格	行末有没有多输出一个多余空格？

相对基础队列模板改了哪里？

1. 队列长度随出列不断缩小，因此每轮取模的模数必须是当前的 `q.size()`，不能写死成 n！
2. 移走前 $m-1$ 个人后，第 m 个人只 `pop()` 不再 `push()`。

例 2 | 完整可编译代码（1/2）：入队与整圈取模

以下源码与 code/circle_count.cpp 完全一致：

```cpp
#include <bits/stdc++.h>
using namespace std;
// 围圈报数（约瑟夫环队列模拟）：n 人围圈每次报到 m 出列，输出出列顺序
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, m;
    if (!(cin >> n >> m)) return 0;
    queue<int> q;
    for (int i = 1; i <= n; ++i) q.push(i);        // 按 1..n 顺序入队
```

本页要点

用 while (!q.empty()) 推进，直到所有人都出列为止。

例 2 | 完整可编译代码（2/2）：轮转、出列与格式输出

```
    bool first = true;
    while (!q.empty()) {
        int step = (m - 1) % (int)q.size();          // 取模跳过绕整圈的多余报数
        for (int i = 0; i < step; ++i) {
            q.push(q.front());                       // 报 1..m-1 的人排回队尾
            q.pop();
        }
        if (!first) cout << ' ';
        cout << q.front();                           // 第 m 个人出列并打印
        first = false; q.pop();
    }
    cout << '\n'; return 0;
}
```

本页要点

注意 `(int)q.size()`：因为 `q.size()` 是无符号类型，强转成 `int` 更安全。

例 2 | 易错点与 3 组自测边界

- 易错 **1**：取模写死成 `(m - 1) % n`，人出列后圈变小了却仍按 n 取模，导致 **WA**。
- 易错 **2**：循环移了 m 次而不是 $m-1$ 次，把第 $m+1$ 个人当成出列者，导致 **WA**。
- 易错 **3**：没取模直接跑 `for (i=0; i<m-1; ++i)`，遇大 m 数据直接 **TLE**。

3 组必跑自测边界

1. 原题样例：`4 17` $\Longrightarrow$ `1 3 4 2`（测试 $m > n$ 跳过整圈）。
2. $m = 1$ 边界：`5 1` $\Longrightarrow$ `1 2 3 4 5`（`step = 0` 直接依次出列）。
3. 单人边界：`1 100` $\Longrightarrow$ `1`（仅 1 人，直接出列）。

例 3 | 奇怪的电梯（1360）[基础]：先读懂要求

大白话题意

大楼共 N 层（$1 \le N \le 200$），第 i 层有个数字 K_i。在 i 层按“上”可以跳到 $i+K_i$ 层，按“下”可以跳到 $i-K_i$ 层；如果目标楼层超出 $1 \sim N$ 范围，按钮就失灵。问从 A 楼到 B 楼至少需要按几次按钮？无法到达输出 -1。

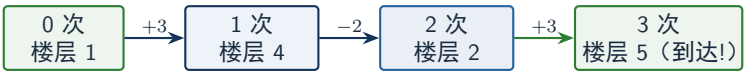
- 输入：第 1 行 N, A, B；第 2 行 N 个整数 $K_1 \dots K_N$。
- 处理：用队列做广度优先搜索（BFS），按“按键次数 $0, 1, 2, \dots$”逐层扩展楼层。
- 输出：一行一个整数，最少按键次数（不可达输出 -1）。

本页要点

原题样例：输入 `5 1 5 / 3 3 1 2 5`，路线 $1 \to 4 \to 2 \to 5$，输出 3。

例 3 | 怎么想到的？为什么“按步数排队”一定是最少按键？

每按一次按钮，步数恰好加 1。用队列 q 按“先到达先入队”的顺序搜（即广度优先搜索 **BFS**）：



- 因为队列先进先出，所有按 0 次到达的楼层排在前面，其次是按 1 次到达的、再其次是按 2 次到达的……
- 所以第一次访问到目标楼层 B 时记录的 `dist[B]`，一定是全局最少按键次数！
- 同时用 `dist[v] == -1` 兼作“是否走过”，走过的楼层绝不再进队（避免死循环绕圈）。

本页要点

边权全为 1 的最短路问题，直接上队列 BFS，首次碰到终点就可以 `break`！

例 3 | 样例手推跟踪表：输入 5 1 5 / 3 3 1 2 5

数组 $jump[1..5] = [3, 3, 1, 2, 5]$。初始 `dist = [-, 0, -1, -1, -1, -1]`，`q = [1]`：

出队 `u`	步数 `dist[u]`	尝试上下 `u ± jump[u]`	合法且未访问的新楼层	操作后队列 `q`
1	0	$1+3=4$，$1-3=-2$（越界）	$v=4 \Longrightarrow$ `dist[4]=1`	`[4]`
4	1	$4+2=6$（越界），$4-2=2$	$v=2 \Longrightarrow$ `dist[2]=2`	`[2]`
2	2	$2+3=5$，$2-3=-1$（越界）	$v=5 \Longrightarrow$ `dist[5]=3`	`[5]`
5	3	发现 `u == b (5 == 5)`！	立刻 `break` 退出	输出 `dist[5] = 3`

本页要点

每层楼最多只入队 1 次，总时间复杂度仅为 $O(N)$！

例 3 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
建 `dist(n+1, -1)`	起点 `dist[a] = 0`	若 $a == b$，是否能正确直接输出 0？
队列弹出队头 `u`	若 `u == b` 则 `break`	越界检查 `v >= 1 && v <= n` 写全了吗？
扩展 `u ± jump[u]`	`dist[v] = dist[u] + 1`	是否在入队时就赋值 `dist[v]` 防重复入队？

相对基础队列模板改了哪里？

1. 用 `dist` 数组同时完成“记录最少次数”和“判重（`-1` 表示未走过）”两件事。
2. 初值全部设 `-1`，如果终点永远不可达，循环结束自然输出 `-1`。

例 3 | 完整可编译代码（1/2）：读入与起点初始化

以下源码与 code/elevator_bfs.cpp 完全一致：

```cpp
#include <bits/stdc++.h>
using namespace std;
// 一本通 1360 奇怪的电梯：队列 BFS 求从 A 楼到 B 楼的最少按键次数
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, a, b;
    if (!(cin >> n >> a >> b)) return 0;
    vector<int> jump(n + 1);
    for (int i = 1; i <= n; ++i) cin >> jump[i];
    vector<int> dist(n + 1, -1);
    queue<int> q;
    dist[a] = 0; q.push(a);
```

本页要点

楼层从 1 到 n 编号，因此数组开 n + 1 大小，直接用 1-based 下标。

例 3 | 完整可编译代码（2/2）：队列 BFS 扩展与输出

```
    while (!q.empty()) {
        int u = q.front(); q.pop();
        if (u == b) break;
        for (int v : {u + jump[u], u - jump[u]}) {
            if (v >= 1 && v <= n && dist[v] == -1) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
    cout << dist[b] << '\n';
    return 0;
}
```

本页要点

必须在 `q.push(v)` 的同时标记 `dist[v]`，不能等出队时才标记，否则同一楼层会重复入队！

例 3 | 易错点与 3 组自测边界

- 易错 **1**：忘了判 `v >= 1 && v <= n`，导致访问 `dist[-2]` 触发 **RE**。
- 易错 **2**：忘了 `dist[v] == -1` 判重，若 1 和 4 能互跳就会无限入队导致 **TLE/MLE**。
- 易错 **3**：特判逻辑把 `a == b` 输出成了 `-1` 而不是 `0`。

3 组必跑自测边界

1. 原题样例：`5 1 5 / 3 3 1 2 5` $\Longrightarrow$ 输出 `3`。
2. 起点即终点：`5 3 3 / 1 1 1 1 1` $\Longrightarrow$ 输出 `0`（测试不用按键）。
3. 不可达边界：`4 1 4 / 2 2 2 2` $\Longrightarrow$ $1 \leftrightarrow 3$ 死锁到不了 4，输出 `-1`。

例 4 | 最敏捷的机器人（1542）[提高]：先读懂要求

大白话题意

有一排共 n 个整数（$1 \le k \le n \le 10^5$），每个数在 int 范围内（$[-2^{31}, 2^{31}-1]$）。请依次求出每一个长度为 k 的连续区间（即 $[1..k], [2..k+1], \ldots, [n-k+1..n]$）中的最大值和最小值。

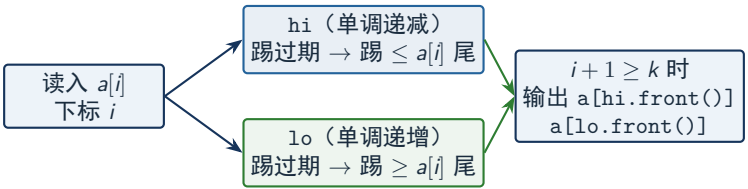
- 输入：第 1 行 n, k；第 2 行 n 个整数。
- 处理：一趟扫描中用两个单调队列分别维护当前窗口的最大值与最小值。
- 输出：共 $n-k+1$ 行，每行两个整数：该窗口的最大值与最小值。

本页要点

原题样例：输入 `5 3 / 1 2 3 4 5`，依次输出 `3 1`、`4 2`、`5 3`。

例 4 | 怎么想到的？一条流水线同时跑“递减队”与“递增队”

本题和单调队列模板唯一的区别是：每个窗口既要最大值，又要最小值。那就并排开两个互不干扰的 deque：



- 两个队列都只存下标 i，且都使用同样的过期条件 `front() <= i - k`。
- 输出行序要求“先最大值、后最小值”，即先打 `a[hi.front()]` 再打 `a[lo.front()]`。

本页要点

两个单调队列各自满足均摊 $O(n)$，合起来总时间依然是绝对稳健的 $O(n)$！

例 4 | 样例手推跟踪表：输入 5 3 / 4 2 5 1 3

为了练一下升降交错的数据，我们手推 $a=[4,2,5,1,3]$，$k=3$：

i ($a[i]$)	hi（递减求最大）	lo（递增求最小）	满 3 项输出（最大最小）
0 (4)	$[0_{(4)}]$	$[0_{(4)}]$	未满
1 (2)	$[0_{(4)}, 1_{(2)}]$	踢 $0_{(4)} \to [1_{(2)}]$	未满
2 (5)	踢 $1, 0 \to [2_{(5)}]$	$[1_{(2)}, 2_{(5)}]$	5 2
3 (1)	过期无 $\to [2_{(5)}, 3_{(1)}]$	过期无，踢 $2, 1 \to [3_{(1)}]$	5 1
4 (3)	过期无 $\to [2_{(5)}, 4_{(3)}]$	过期无 $\to [3_{(1)}, 4_{(3)}]$	5 1

本页要点

注意 $i=2$ 时 5 把 hi 里的 $2, 4$ 全踢走，却老实排在 lo 的 2 后面；两个队列各踢各的，互不干扰！

例 4 | 核心步骤与相对模板改动

按顺序计算	纸上得到	核对问题
建 `hi`、`lo` 两个双端队列	两者均存下标 `i`	两个队列的过期条件都是 `<= i - k` 吗？
同步清理队头与队尾	`hi` 踢 `<= a[i]`；`lo` 踢 `>= a[i]`	`lo` 的不等号方向有没有不小心写反？
满 k 项后成对输出	`a[hi.front()]` 与 `a[lo.front()]`	题目要求“先最大再最小”，顺序对吗？

相对基础单调队列模板改了哪里？

1. 开了第二个队列 `lo`，仅把队尾比较方向从 `<=` 改成 `>=`。
2. 每行输出 2 个数，共 $n-k+1 \le 10^5$ 行，必须用 `'\n'` 换行并关闭同步流防 TLE。

例 4 | 完整可编译代码（1/2）：读入与双队列准备

以下源码与 code/agile_robot.cpp 完全一致：

```cpp
#include <bits/stdc++.h>
using namespace std;
// 一本通 1542 最敏捷的机器人：双单调队列 O(n) 求定长 K 滑动窗口的最大值与最小值
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, k;
    if (!(cin >> n >> k)) return 0;
    vector<int> a(n);
    deque<int> hi, lo;                                  // 队列里存的是下标，不是元素值！
```

本页要点

数组 a(n) 用 0-based 下标，对应窗口满 k 个的条件是 i + 1 >= k。

例 4 | 完整可编译代码（2/2）：双单调队列主循环与输出

```
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
        while (!hi.empty() && hi.front() <= i - k) hi.pop_front(); // 1. 队头滑出窗口先踢掉
        while (!lo.empty() && lo.front() <= i - k) lo.pop_front();
        while (!hi.empty() && a[hi.back()] <= a[i]) hi.pop_back(); // 2. 踢掉不如新元素且早过期的队尾
        while (!lo.empty() && a[lo.back()] >= a[i]) lo.pop_back();
        hi.push_back(i); lo.push_back(i);           // 3. 当前下标入队尾
        if (i + 1 >= k) cout << a[hi.front()] << ' ' << a[lo.front()] << '\n';
    }
    return 0;
}
```

本页要点

代码总共只有 23 行，就能在 $O(n)$ 时间内同时搞定 10 万长度窗口的最大和最小值！

例 4 | 易错点与 3 组自测边界

- 易错 **1**：输出时把顺序写成先最小后最大（`lo` 在前 `hi` 在后），导致 **WA**。
- 易错 **2**：输出换行写了 `endl`，在 10^5 行输出时频繁刷新缓冲区导致 **TLE**。
- 易错 **3**：只踢了 `hi` 的过期队头，忘了同步踢 `lo` 的过期队头。

3 组必跑自测边界

1. 原题样例：`5 3 / 1 2 3 4 5` $\Longrightarrow$ `3 1`、`4 2`、`5 3`。
2. $k = 1$ 边界：`3 1 / 7 -2 5` $\Longrightarrow$ `7 7`、`-2 -2`、`5 5`（最大最小相等）。
3. $k = n$ 边界：`4 4 / 9 2 7 4` $\Longrightarrow$ 仅输出 1 行 `9 2`。

队列与单调队列高频错误对照与定位指南

题目 / 场景	常见错误写法	正确做法与原因
围圈报数（1334）	直接循环 $m-1$ 次不取模	遇大 m 超时 **TLE**，改用 `(m-1) % q.size()`
单调队列存元素值	把 `a[i]` 塞进 deque	无法识别队头元素是否已滑出左窗口，必须存下标 `i`
出界检查写错条件	写 `hi.front() < i - k`	窗口 $[i-k+1, i]$ 下界外最后一个失效位置是 `i - k`，写 `<= i - k`
提前输出未满窗口	没加 `if (i + 1 >= k)`	前 $k-1$ 步窗口长度还不足 k，绝不能提前打印
空队列取队头	未判 `!q.empty()` 直接 `front/pop`	访问空指针导致段错误 **RE**

- 定位口诀：**WA** 80% 是出界判断把 `<= i - k` 错写成 `<` 或提前输出了未满窗口；**RE** 查空队列与 1-based 下标越界。

本页要点

单调队列永远记住四步：“存下标、踢过期、去弱尾、满 k 再出答案”！

提交前检查清单（六必查）

1. 单调队列中保存的是不是元素下标，并用 `a[dq.back()]` 对比元素大小？
2. 每轮循环第一步是否判断了 `dq.front() <= i - k`（若是 1-based 则同为 `<= i - k`）及时弹出过期队头？
3. 是否分别把最大值（剔掉 `<= a[i]`）和最小值（剔掉 `>= a[i]`）的比较方向写对？
4. 是否确保只有当 `i + 1 >= k`（0-based）或 `i >= k`（1-based）时才开始输出窗口极值？
5. 普通 BFS/循环队列访问 `front()` 前有没有判断 `!q.empty()`，并处理无法到达输出 `-1`？
6. 大量行输出（如 10^5 行滑动最值）时，有没有禁用同步流并使用 `'\n'`？

最后一步

造一组 $k = 1$（窗口宽度为 1）、严格递减序列（测试窗口滑出踢头）和全相等序列测试后再交！

本章全部练习与完成标准

1. 普通队列与排队：周末舞会（一本通 1332，`code/weekend_dance.cpp`）、围圈报数（一本通 1334，`code/circle_count.cpp`）。
2. 队列 **BFS** 最短路：奇怪的电梯（一本通 1360，`code/elevator_bfs.cpp`）。
3. 单调队列实战：最敏捷的机器人（一本通 1542，`code/agile_robot.cpp`）。

完成标准

能口述为什么滑动窗口可以用 $O(n)$ 单调队列代替堆；能一次写对带双向过期踢头的双端队列代码；独立 AC 上述精选习题。

课后安排与下一章预告

- 复习：在纸上走完一遍 $a = [1, 3, -1, -3, 5, 3], k = 3$ 的单调队列表格，注意看 $i = 4$ 时队头下标 1 是怎么被滑掉的。
- 练习：用单调队列独立重写《最敏捷的机器人》并与 $O(nk)$ 暴力对照随机数据拍验 100 组。
- 下一章预告：第 3 章《树、二叉树与堆》——正式离开一维数组和线性队列，踏入树形层次结构、前中后序遍历与二叉堆贪心合并！

本章一句话

普通队列保先来后到的公平，单调队列保窗口最值的 $O(n)$ 极速。

谢谢！