单 调 栈 和 单 调 队 列

衡 水 第 一 中 学

信 息 奥 赛

第　一　节

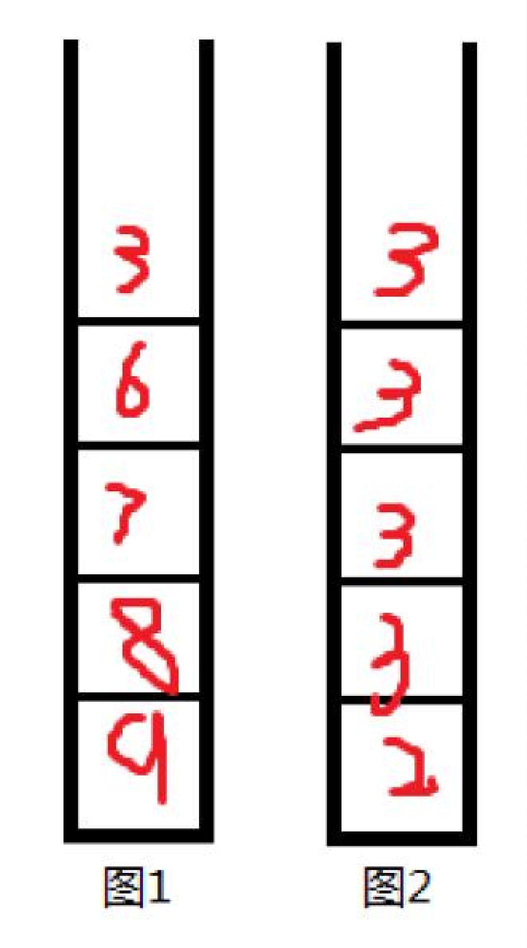
单调栈

什么是单调栈

单调栈是指一个栈内部的元素具有严格单调性的一种数据结构，分为单调递增栈和单调递减栈。

单调栈的性质

1. 满足栈底到栈顶的元素具有严格单调性。
2. 满足栈的先进后出特性，越靠近栈顶的元素越后出栈。



如图所示：
图1 所示栈为单调递减栈
图2 所示栈既不是单调递增栈，也不是单调递减栈

元素进栈过程

对于一个单调递减栈来说，若当前进栈的元素为a，如果 a<栈顶元素，则直接将a进栈。
如果 a≥栈顶元素，则不断将栈顶元素出栈，直到满足 a<栈顶元素或栈为空，再将 a 入栈。

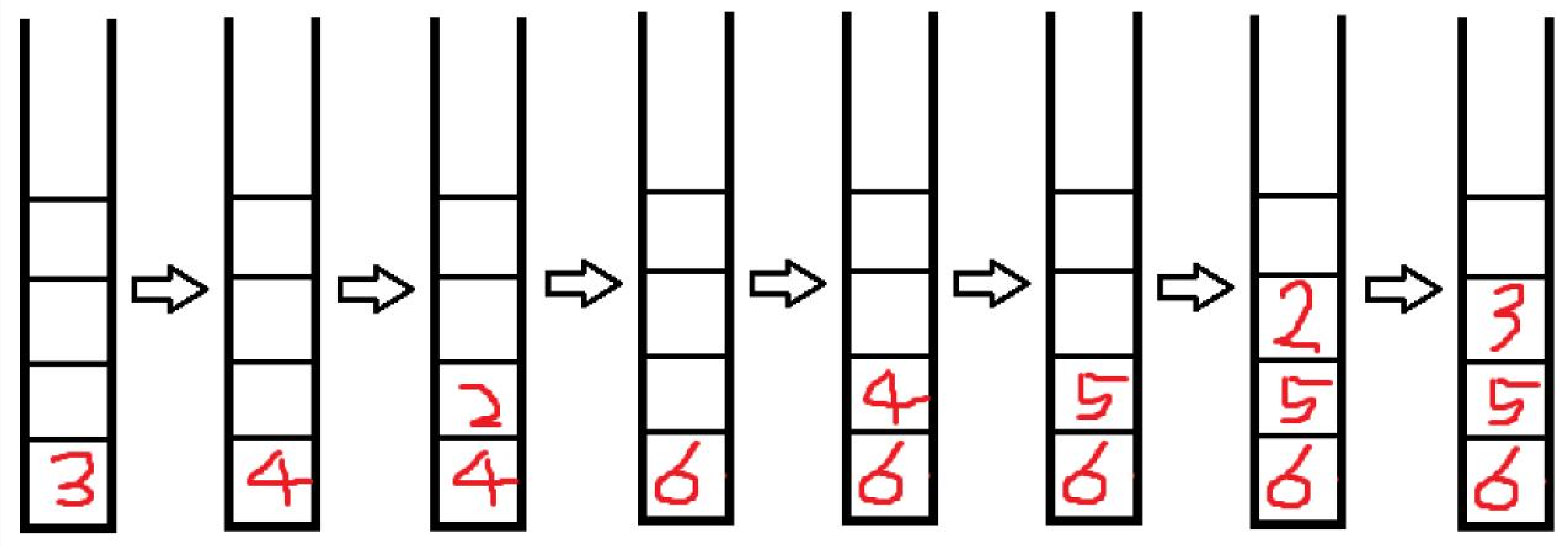
实现单调栈

实现单调栈STL栈和手写栈均可。

模拟一个数列构造一个单调递减栈

进栈元素分别为3，4，2，6，4，5，2，3。

图片所示过程即为进栈过程。



应用例题一

广告印刷，求某个点根据高度向左向右延伸的长度

应用例题二

用于离线解决 RMQ （区间最值查询）问题。

我们可以把所有询问[li, ri]按右端点排序，然后每次在序列上从左往右扫描到当前询问的右端点处，并把扫描到的元素插入到单调栈中。这样，每次回答询问时，单调栈中存储的值都是位置<=ri的、可能成为答案的决策点，并且这些元素满足单调性质。此时，单调栈上第一个位置>=li的元素就是当前询问的答案，这个过程可以用二分查找实现。使用单调栈解决 RMQ 问题的时间复杂度为 O(qlogq+qlogn)，即对询问排序+对每个询问的二分查找，空间复杂度为 O(n)。

第　　二　　节

单调队列

什么是单调队列

单调队列与单调栈及其相似，把单调栈先进后出的性质改为先进先出即可。

单调递增队列的元素进队过程

对于一个元素a，如果 $a >$ 队尾元素，那么直接将a扔进队列，如果$a \le$ 队尾元素，则将队尾元素出队列，直到满足 $a >$ 队尾元素或队列为空，再将a从队尾入队。

同时，如果队首元素，已经失效，则队首元素出队，直到队首元素仍在有效范围内为止。

实现单调队列

实现单调队列使用STL的deque（双端队列）即可。

由于双端队列即可以在队头操作，也可以在队尾操作，
那么这样的性质就弥补了单调栈只能在一边操作的不足。
可以使得其队首也有一定的限制。

应用例题一——滑动窗口

给你n个数，让你在这n个数中选出连续的m个数（$m \le n$），使这m个数的极差最小，若存在多个区间使得极差均最小，输出最靠前的区间。

很显然，$n \le 10^4$ 时暴力明显可做，$n \le 10^6$ 时通过线段树也可做，那如果 n 去到 $n \le 10^7$ 呢？

我们考虑用单调队列维护区间的最小值和最大值，以下篇幅以维护最小值举例。

应用例题一——滑动窗口

首先，我们把前m个数扔进一个单调递增队列中，在扔进去的同时把这些数所对应的下标也扔进去。显然队首的数字即为区间[1,m] 最小的数。

考虑基于[1,m]的数据去更新[2,m+1]的最小值。若第一个数（下标为1的数）依然存在于队列中（很显然若存在仅可能位于队首），将这个数从队首删除，然后将第m+1个数插入该单调队列的队尾，同时维护队列的单调性。显然队首数字即为区间[2,m+1]的最小值。

重复该过程n-m+1次即可，时间复杂度为O(n)。

应用例题——滑动窗口

例如序列：1 3 -1 -3 5 3 6 7，连续区间m=3，取最小值，算法过程如下图所示：

操作	队列状态
1 入队	{1}
3 比 1 大，3 入队	{1 3}
-1 比队列中所有元素小，所以清空队列 -1 入队	{-1}
-3 比队列中所有元素小，所以清空队列 -3 入队	{-3}
5 比 -3 大，直接入队	{-3 5}
3 比 5 小，5 出队，3 入队	{-3 3}
-3 已经在窗体外，所以 -3 出队；6 比 3 大，6 入队	{3 6}
7 比 6 大，7 入队	{3 6 7}

应用例题———滑动窗口

参考代码：

```
// 得到这个队列里的最小值，直接找到最后的就行了
// 区间设定为k，队列里存的是原序列的下标
void getmin() {
  int head = 0, tail = 0;
  // 先把前k-1个数放入队列
  for (int i = 1; i < k; i++) {
    // 队尾插入元素时，维护单调性
    while (head <= tail && a[q[tail]] >= a[i]) tail--;
    q[++tail] = i;
  }
  // 从第k个数开始，每次加入一个数，输出一个最值
  for (int i = k; i <= n; i++) {
    while (head <= tail && a[q[tail]] >= a[i]) tail--;
    q[++tail] = i;
    // 已经失效的队首的元素出队
    while (q[head] <= i - k) head++;
    // 队首元素为最小值
    printf("%d ", a[q[head]]);
  }
}
```

应用例题二——Luogu P2698 Flowerpot S

老板需要你帮忙浇花。给出N滴水的坐标，y表示水滴的高度，x表示它下落到x轴的位置。

每滴水以每秒1个单位长度的速度下落。你需要把花盆放在x轴上的某个位置，使得从被花盆接着的第1滴水开始，到被花盆接着的最后1滴水结束，之间的时间差至少为D。

我们认为，只要水滴落到x轴上，与花盆的边沿对齐，就认为被接住。给出N滴水的坐标和D的大小，请算出最小的花盆的宽度W。

应用例题二——Luogu P2698 Flowerpot S

题意可以转化为求一个 x 坐标差最小的区间使得这个区间内 y 坐标的最大值和最小值之差至少为 D。

我们发现这道题和上一道例题有相似之处，都与一个区间内的最大值最小值有关，但是这道题区间的大小不确定，而且区间大小本身还是我们要求的答案。

应用例题二——Luogu P2698 Flowerpot S

设[L, R]为我们要求的x坐标的区间

我们依然可以使用一个递增，一个递减两个单调队列在 R 不断后移时维护 $[L, R]$ 内的 y 坐标的最大值和最小值。

不过此时我们发现，如果 L 固定，随着 R 的不断后移，$[L, R]$ 内的最大值只会越来越大，最小值只会越来越小。所以设 $f(R)=max[L, R]-min[L, R]$（这里的max、min表示区间 $[L, R]$ 内的最大小值），则 $f(R)$ 是个关于 R 的递增函数，故 $f(R) \geq D$ ➔$f(r)>D, R<r\leq N$，也就是说，如果此时我们固定了花盆的左边界，此时找到花盆一个右边界 R 满足条件，那么任何一个大于 R 的值也必定满足条件。

这说明对于每个固定的 L，向右第一个满足条件的 R 就对应最优答案。

应用例题二——Luogu P2698 Flowerpot S

所以我们整体求解的过程就是：

先固定 L，从前往后移动 R，使用两个单调队列维护 $[L, R]$ 的最值。

当找到了第一个满足条件的 R（即 y 坐标的最大差值 $\geq D$），就更新答案，并将 L 也向后移动同时继续更新答案。

随着 L 向后移动，两个单调队列都需及时弹出队头（L 后移意味着花盆左边界右移，原来队首的失效的元素要出队），再重复上面的操作找最优解。

这样，直到 R 移到最后，每个元素依然是各进出队列一次，保证了 $O(n)$ 的时间复杂度。

应用例题二——Luogu P2698 Flowerpot S

参考代码

```
sort(a + 1, a + n + 1);
hx = hn = 1; // 两个队列队首指针
rx = rn = 0; // 两个队列队尾指针
ans = 2e9;
int L = 1; // 左边界初始化
for (int i = 1; i <= n; ++i) { // 循环左边界
  while (hx <= rx && a[mxq[rx]].y <= a[i].y) rx--;
  mxq[++rx] = i; // 最大值的队列
  while (hn <= rn && a[mnq[rn]].y >= a[i].y) rn--;
  mnq[++rn] = i; // 最小值的队列
  // 判断满足条件，更新答案
  // 找到一个R，可能会让L多次右移
  while (L <= i && a[mxq[hx]].y - a[mnq[hn]].y >= D) {
    ans = min(ans, a[i].x - a[L].x);
    L++;
    while (hx <= rx && mxq[hx] < L) hx++;
    while (hn <= rn && mnq[hn] < L) hn++;
  }
}
```