马上开始

35tang-C++竞赛系列四阶课程

三五堂

《35tang-C++竞赛系列四阶课程》

数组的双指针

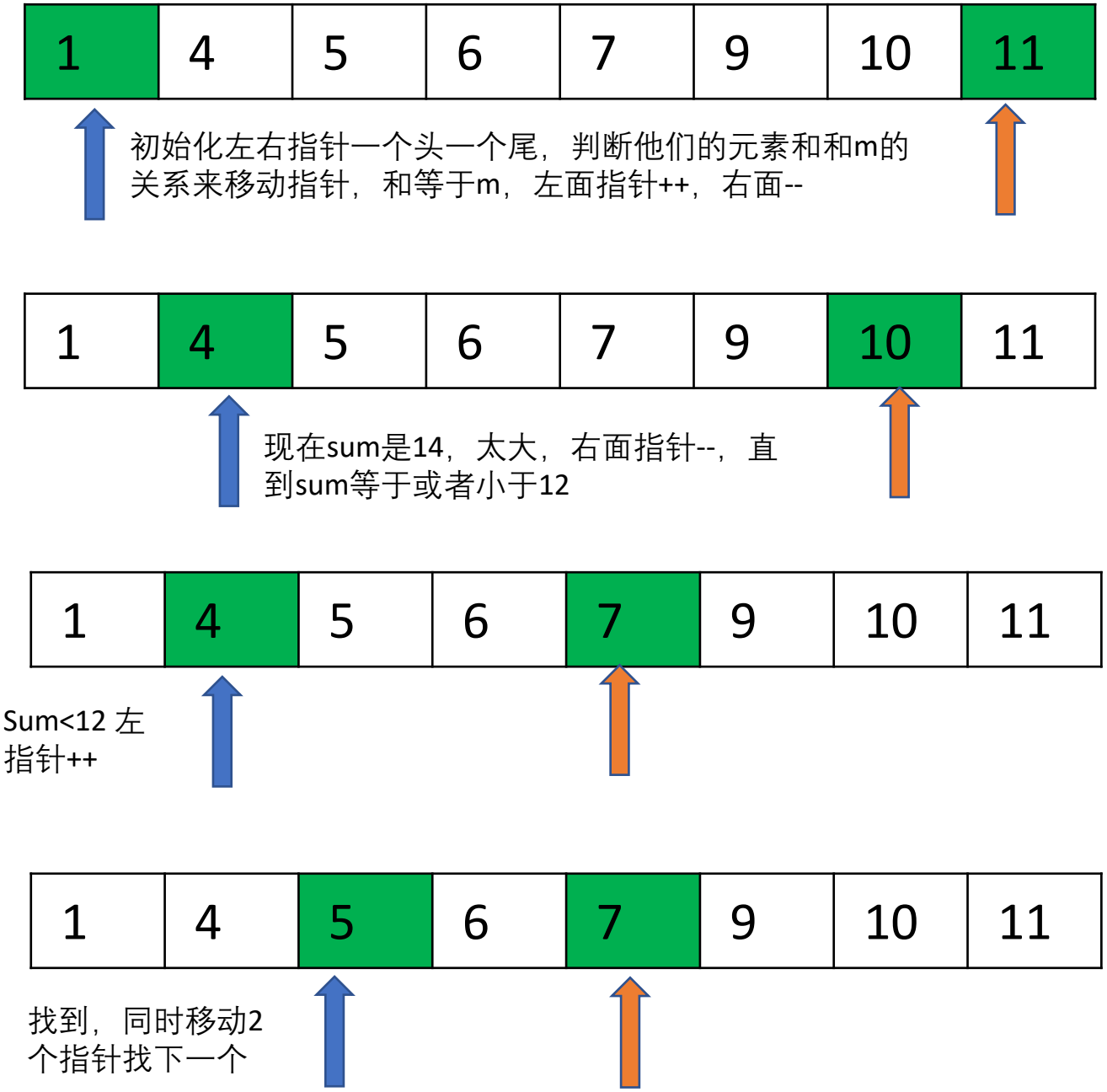
- 一般遍历数组用数组下标的循环，这就是一种指针，因为数组下标指向了数组的每一个位置，找到每一个位置的元素。
- 有时候可以用两个指针（数组下标）在一个范围内查找，两个指针一前一后，相当于限定了一个范围，这样就可以避免一些重复的遍历
- 通过例子来学习

数字和

- 给定n个整数（不重复）的序列，寻找和为m的元素对的个数。
- 例如{1,4,5,6,7,9,10,11} m=12，显然1 11，5 7可以，2对。
- 双循环枚举所有数字对组合，可以，时间复杂度O(n*n)
- 方法1：前面讲过，排序，然后循环选择数组第一个元素a[i]作为元素对的第一个，通过二分去查找第二个元素m-a[i]。
- 方法2：排序后双指针，时间复杂度O(n*logn)，主要时间消耗在排序上。

先排序，然后双指针，一个头一个尾，加起来大于m就右指针--，否则左指针++

循环条件(l<r)



```cpp
#include <bits/stdc++.h>
using namespace std;
int a[100001];
int n,m;
int ans;
int main()
{

    cin>>n>>m;
    for (int i=1;i<=n;i++) cin>>a[i];
    sort(a+1,a+1+n);

    int l=1,r=n;//左右指针
    while (l<r)
    {
        if (a[l]+a[r]==m) {
            ans++;l++;r--;
        }
        else if  (a[l]+a[r]<m) l++;
        else r--;
    }
    cout<<ans<<endl;
    return 0;
}
```

双指针

while循环

注意if else嵌套分了3种情况

```
int l=1,r=n;
while (l<r)
{
        if (a[l]+a[r]==m) {
                ans++;l++;r--;
        }
        else if  (a[l]+a[r]<m) l++;
        else r--;
}
```

改写成for循环

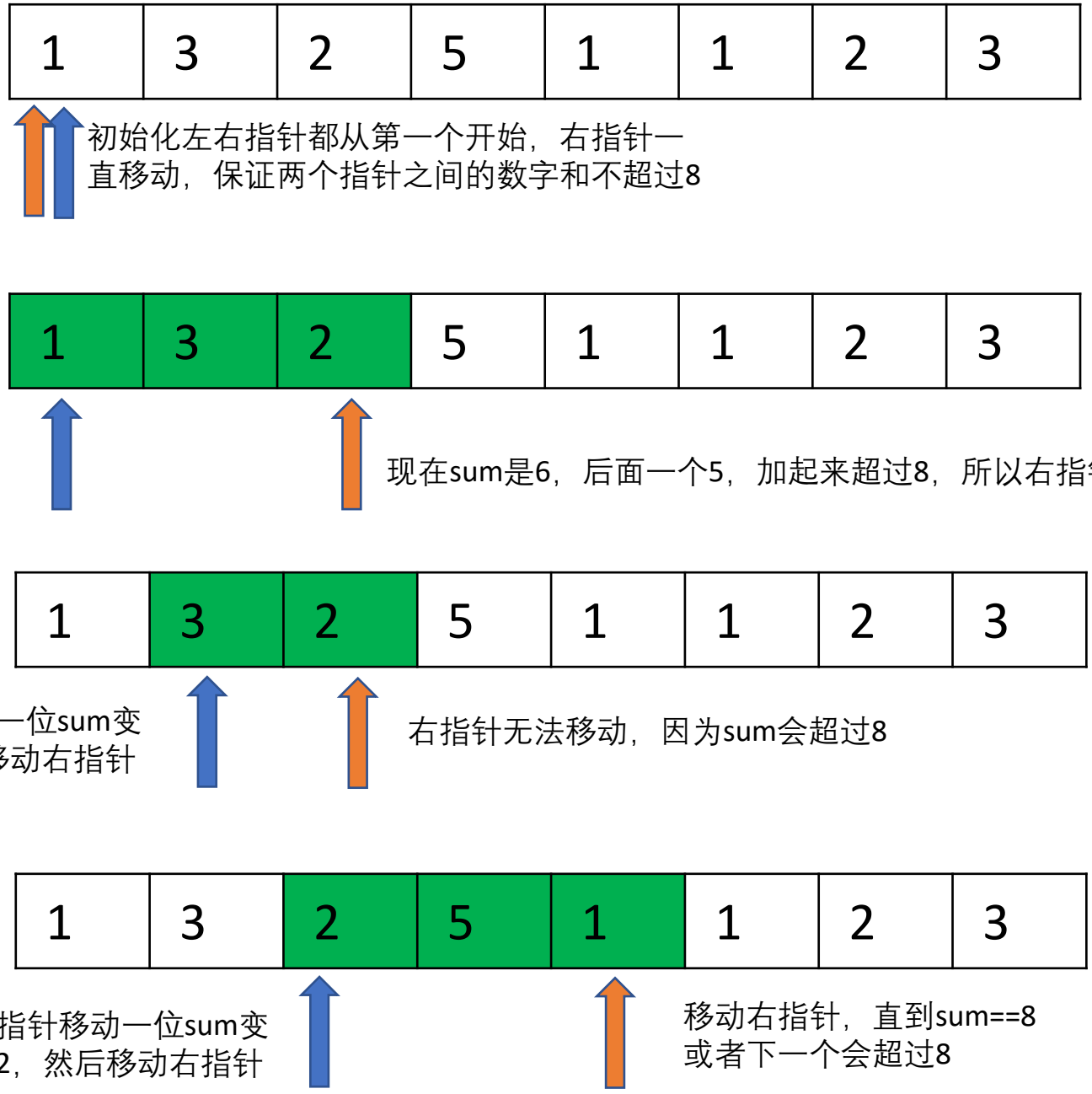
逻辑是一样的

```
for(int i=1;i<=n;++i)
{
        for(int r=n;r>i;r--)
        {
                if(a[i]+a[r]==m)
                        ans++;
                if(a[i]+a[r]<=m)
                        break;
        }
}
```

连续子序列和

- 给出一个正整数序列，统计连续子序列的和是m的有多少个。例如给定数字序列{1,3,2,5,1,1,2,3}求m是8的连续子序列。显然，只有2 5 1加起来是8，所以答案是1.有1个连续子序列和是8.
- 连续子序列，所以不能排序
- 前面讲过很多方法，这里再看看双指针的方法。
- 思路很简单，挨个尝试这个连续子序列的第一个元素的位置（左指针），往右找和为m的子序列，每次固定开始位置之后，把后面所有元素都加起来直到超过m？这里明显有重复的部分（开始位置i和i+1往右找肯定重复计算了好多元素），能不能维护一个当前连续子序列的和sum，然后每一次开始位置往右移动的时侯就只要给这个sum减去最左面的元素，而不需要重新计算右面已经计算过的所有元素的和。例如：1 3 2 和为6，找不到，接下来找3开始的，这时候和就变成了6-1=5，再接着刚才找到的2这个元素的位置往右找。

用两个指针维护一个滑动窗口以及当前窗口的数字和sum。固定左指针，尝试移动右指针找和为sum，找不到了就移动左指针。左右指针维护了一个和为sum的移动窗口。



```cpp
#include <bits/stdc++.h>
using namespace std;
int a[200001];
int n,m;
int ans,sum;
int main()
{

    cin>>n>>m;
    for (int i=1;i<=n;i++) cin>>a[i];

    int l=1,r=1;//左右指针
    sum=a[1];
    while (l<=n)
    {
        while (r<n && sum+a[r+1]<=m) {r++;sum+=a[r];}
        if (sum==m) ans++;
        sum-=a[l];l++;
    }
    cout<<ans<<endl;
    return 0;
}
```

思考：如果没有r<n会如何？

```
int l=1,r=1;//左右指针
sum=a[1];
while (l<=n)
{
        while (r<n && sum+a[r+1]<=m)
                {r++;sum+=a[r];}
        if (sum==m) ans++;
        sum-=a[l];
        l++;
}
cout<<ans<<endl;
```

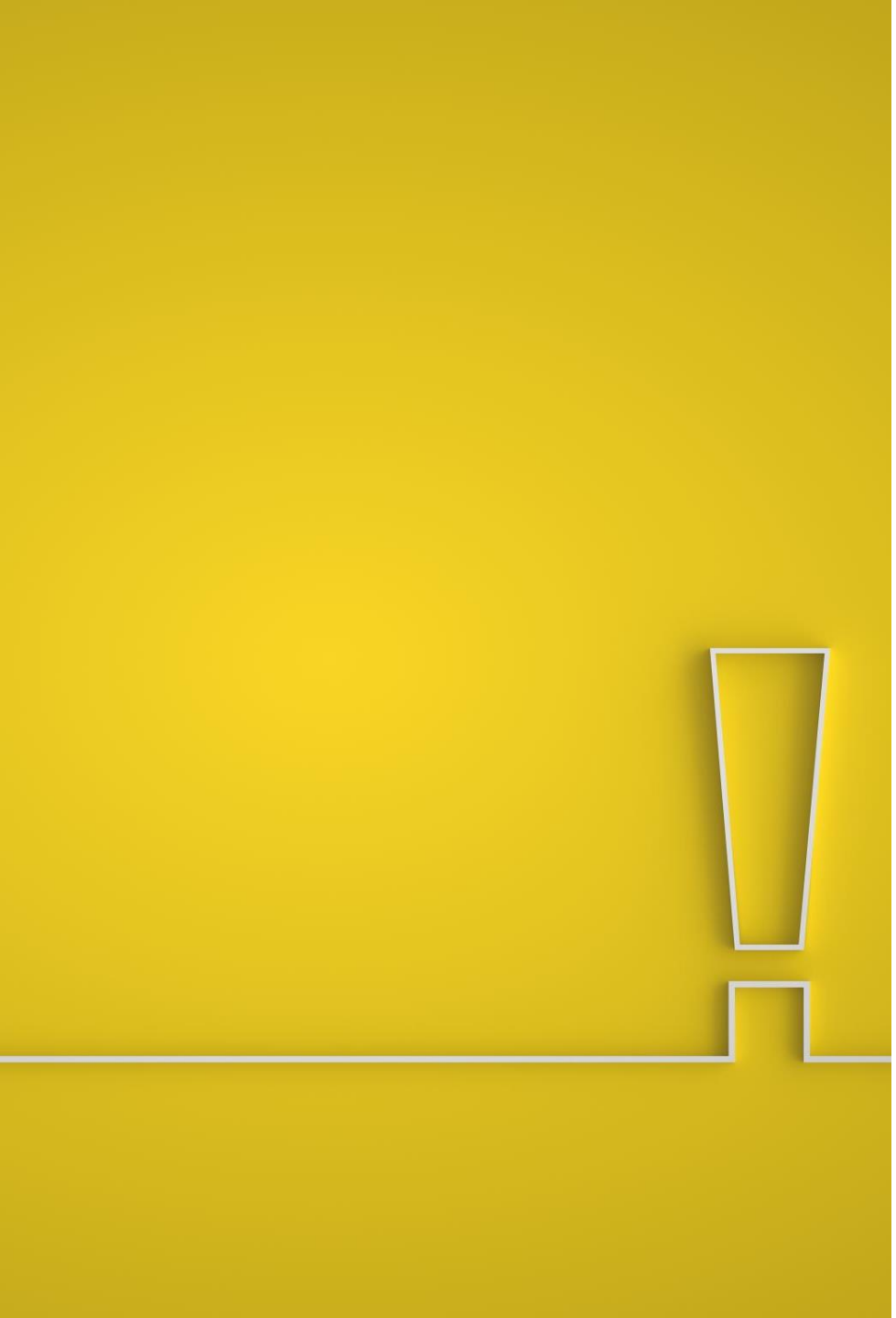
单调队列

- 单调递减或单调递增的队列。不断地向缓存数组里读入元素，也不时地去掉最老的元素，不定期的询问当前缓存数组里的最小（最大）的元素。用单调队列来解决问题，**一般都是需要得到当前的某个范围内的最小值或最大值。**
- 举个例子：有 7 6 8 15 9 10 4 七个数字，找出范围（i-4，i） 的最小值。
- 最简单的办法，对于每一个i遍历他前面的4个元素，找最小值。
- 如果数组元素个数n很大，每一次寻找i-k个元素，时间复杂度就是O(n*k)。能否优化？这里我们发现计算i+1的时侯，其实前面至少有k-1个元素是上一次计算i的时侯计算过的。

7 6 8 15 9 10 4　（数组从1开始编号）

单调队列长度不超过4，队列从小到大排列，队头就是每一次的最小：每一次新元素从队列尾部往前找，比当前新元素大的出队（因为没有必要保留比当前元素远并且大的）；再从队头往后找，距离超过4的出队。每一个位置的答案就是当前单调队列的头。

i从1到n循环	当前队列	当前元素	操作	之后队列
i=1		7	7入队列	7
i=2	7	6	7出队，6入队	6
i=3	6	8	8入队列	6 8
i=4	6 8	15	15入队列	6 8 15
i=5	6 8 15	9	15出队，9入队列	6 8 9
i=6	6 8 9	10	10入队列，6出队(距离太大)	8 9 10
i=7	8 9 10	4	4入队列,其他全部出（比4大的没有必要留着）	4



单调队列的要点

- 不需要也无法先排序,遍历数组，对每一个位置都维护一个元素个数不超过k的队列，如果找前k个最小，实际上这个队列里面保存的就是当前i元素前k个元素里面比i元素小的所有数和各自的位置，而且是从小到到大排序的。
- 如果找最近一段的最小，对于每一个元素，前面的比他大的就没有必要保留，这样的化保留下来的队列里面的数据肯定是从小到大排列的，因为前面的一定是小的，大的不要。反过来：由于队列是单调的，所以从队列尾部往前找，不需要遍历队列所有元素就可以找到所有比当前元素大的删除。
- 单调队列一般有距离限制，前面太远的不要。由于插入新元素是从队列后面插入，所以队列前面的一定是距离远的，所以删除距离太远的元素的时侯，从队列头部往后找，也不需要遍历队列所有元素就可以把距离过大的都找出来删除。

单调队列练习

- k最小数

输入n (1<=n<=10000)个数字组成的数字序列，输入格式为第一个数字n，k，后面n个整数。

判断每一个元素是否是他前面k个之内（包含这个元素）的元素中最小值，如果最小值是这个元素，这个元素叫做k最小数，最后输出一共多少个k最小数。

示例输入：

5 3

12 8 7 12 9

示例输出：

3

示例分析：

从前往后，12 8 7 都是前3个元素之内最小的。

- 分析：最简单的枚举，每一个位置i打擂他前面的k个元素，找最小，时间复杂度O(n*k);
- 单调队列：时间复杂度O(n). 遍历每一个位置的同时，维护一个长度不超过k的队列，该队列应该是单调递增的。用双端队列或者数组都可以。

模板程序

注意:

数组实现队列关于l

和r的控制。

l++左面出队

r—右面出队

l>r空队列

初始化l=1,r=0

```cpp
#include <bits/stdc++.h>
using namespace std;
int n,k;  int a[10001]; int l,r, ans;
pair<int,int> myq[10001];//单调队列：第一个是值，第二个是在原数组的位置
int main(){
    cin>>n>>k;
    for (int i=1;i<=n;i++) cin>>a[i];
    l=1;r=0;
    for(int i=1;i<=n;i++){
        //从右往左，超过当前值的出队，因为要保留小的
        while (l<=r&& myq[r].first>a[i]) r--;
        //新元素插入最右面
        r++;
        myq[r]=make_pair(a[i],i);
        //下面删除队列左面距离超过k的
        while (l<=r&& myq[l].second<=i-k) l++;
        //找到最小并且判断是否需要++
        if (myq[l].first==a[i]) ans++;
    }
    cout<<ans<<endl;
    return 0;
}
```

最大的作用把时间复杂度从O(K)减为O(1)

1

一个移动的窗口（或者说队列，大小不超过K，表示范围）。

特别使用遍历序列的时候，对于每一个位置都要找附近K范围的最大最小。

如果没有单调队列，传统方法对于每一个位置，可能都需要从前往后遍历他周围的K个元素，如果K很大？

如果用优先队列？无法控制超个数保持在k以内，因为优先队列可以取出最小，但是无法在O(logK)的时间复杂度内删除最先进来的距离超过K的元素。

可以用双端队列deque，从前面删除，后面添加。

改写 deque

```cpp
#include <bits/stdc++.h>
using namespace std;
int n,k;int a[10001]; int ans;
 //pair<int,int> myq[10001];//单调队列，第一个是值，第二个是在原数组的位置
deque < pair<int,int> > myq;
int main(){
     cin>>n>>k;
     for (int i=1;i<=n;i++) cin>>a[i];
     //l=1;r=0;
     for(int i=1;i<=n;i++)   {
          //插入最右面
          //while (l<=r&& myq[r].first>a[i]) r--;
          while (!myq.empty()&& myq.back().first>a[i])  myq.pop_back();
          //r++;    myq[r]=make_pair(a[i],i);
          myq.push_back(make_pair(a[i],i));
          //删除左面超过k的
          //while (l<=r&& myq[l].second<=i-k) l++;
          while (!myq.empty() && myq.front().second<=i-k) myq.pop_front();
          if (myq.front().first==a[i]) ans++; //找到最小并且判断是否需要++
     }
     cout<<ans<<endl;
     return 0;
}
```

单调队列例题：USACO 2013 November Contest, Silver Crowded Cows

https://www.luogu.com.cn/problem/P3088

- 先看题，理解题目
- 用测试数据验证你是否真的理解了
- 示例输入：

6 4
10 3
6 2
5 3
9 7
3 6
11 2

示例输出

2

分析题目

- 最简单的思路，奶牛按照坐标位置排好序。每一个奶牛找他左右D距离内最大的一个奶牛是否超过他的2倍。
- 怎么找？如果简单枚举，就是奶牛按照位置排序，然后对于每一个奶牛，找他左面和右面距离不超过D的所有奶牛的最大高度。特别符合单调队列，对每一个元素在一定范围（距离不超过D）寻找最大值（高度最大的）。对每一个奶牛可以检查他的左右两个单调队列的最大值是否超过他的两倍。
- 单调队列左右各需要一个，可以走两次循环分别去找左面和找右面，两次算法类似，但是避免放在一起复杂化。
- 单调队列分别是左面和右面距离不超过D的从大到小的数组：超过当前距离的或者比当前高度小的没有必要留着。

按照位置排序

```
fin>>n>>d;
for (int i=1;i<=n;i++)

//数组元素包含每一个奶牛的位置和高度
fin>>aliens[i].x>>aliens[i].h;



//按照位置排序
sort(aliens+1,aliens+n+1,mycmp);
```

队列的维护

l++左面出队

r—右面出队

l>r空队列

初始化l=1,r=0

- 对于每一个方向，都是单调队列。注意，这里维护的单调队列是距离不超过D的元素。而不是前面第几个元素。逻辑是类似的。源数组是按照距离排序的。

```
//找左面有没有
    l=1;r=0;
    for (int i=1;i<=n;i++)
    {
        //从r到l，把高度小于i的高度的删除，也就是r--，
        while (r>=l&& myq[r].h<aliens[i].h) r--;
        //把牛i放到尾巴,保证队列从大到小
        myq[++r]=aliens[i];
        //维护d区间，l++，把位置超过d的删除
        while (r>=l&& myq[l].x+d<aliens[i].x) l++;
        //判断当前最大是否满足条件
        if (myq[l].h>=2*aliens[i].h) flagl[i]=1;
    }
```

队列的维护

l++左面出队

r—右面出队

l>r空队列

初始化l=1,r=0

```
//找右面有没有
l=1;r=0;
//和找左面相比较，只是i循环顺序不同，单调队列维护几乎一样的
for (int i=n;i>=1;i--)
{
    //从r到l，把高度小于i的高度的删除，也就是r--
    while (r>=l&& myq[r].h<aliens[i].h) r--;
    //把i放到尾巴，保证队列从大到小
    myq[++r]=aliens[i];
    //维护d区间，l++，把超过d的删除，这里和找左面不一样
    while (r>=l&& myq[l].x-d>aliens[i].x) l++;
    //判断当前最大是否满足条件
    if (myq[l].h>=2*aliens[i].h) flagr[i]=1;
}

int ans=0;//如果一个位置前面两个方向都标记过了计数器++
for (int i=1;i<=n;i++) if (flagl[i]&& flagr[i]) ans++;
fout<<ans<<endl;
```

小tips

- C++另外一种读写文件的方式

```
#include <stdio.h>
freopen(“test.in", "r", stdin);
freopen(“test.out", "w", stdout);
cin>>
cout<<
```

相当于重新定义了cin,cout，这样后面读写文件可以使用cin cout。

作业1

求相等数对(selectpair.cpp)

给定n个整数的序列（整数可能相同），寻找和为m的元素对。如果没有找到，输出“IMPOSSIBLE”，如果找到，从小到大输出第一对元素的2个位置，空格分开，第一对的意思是指找到的所有数对中较小数最小的那一对数。m和这些整数都是小于10的9次方的自然数。$1 \le n \le 2 \cdot 10^5$

示例输入：

8 12

11 4 5 6 7 9 10 1

示例输出：

1 8

作业2
dandiao.cpp

单调队列练习

k最大数

输入n (1<=n<=10000)个数字组成的数字序列，输入格式位第一个数字n，k，后面n个整数。

判断每一个元素是否是他前面k个之内（包含这个元素）的元素中最大值，如果是这个数叫做k最大数，最后输出一共多少个k最大数。

示例输入:

5 3

12 8 7 12 9

示例输出:

2

作业3
paln.cpp

给定n（n<=1000000）个数字（小于10的9次方）组成的序列，将数列中，任意两个相邻的数字合并，用它们的和来代替，合并完成的值还可以和其他值不断合并，直到只剩下一个数。最后总能得到一个回文序列。一个数肯定是回文数列。求经过最少多少次合并，可以构成一个回文数列。

样例输入

5

1 2 4 6 1

样例输出1

1

由易到难，思维体系训练
实战结合，创新协作培养
兴趣导向，未来职业引领

https://www.35tang.com

https://www.三五堂.com

扫码关注公众号

添加辅导老师