尺取法（又称为：双指针、two pointers），是算法竞赛中一个常用的优化技巧，用来解决序列的区间问题，操作简单、容易编程。

如果区间是单调的，也常常用二分法来求解，所以很多问题用尺取法和二分法都行。

另外，尺取法的的操作过程和分治算法的步骤很相似，有时候也用在分治中。

1 尺取法的概念

什么是尺取法？为什么尺取法能优化呢？

考虑下面的应用背景：

（1）给定一个序列。有时候需要它是有序的，先排序。

（2）问题和序列的区间有关，且需要操作2个变量，可以用两个下标（指针）i、j扫描区间。

对于上面的应用，一般的做法，是用i、j分别扫描区间，有两重循环，复杂度O(n2)。以反向扫描（即i、j方向相反，后文有解释）为例，代码是：

```
for(int i = 0; i < n; i++)            //i从头扫到尾
    for(int j = n-1; j >= 0; j--){    //j从尾扫到头
        ......
    }
1234
```

下面用尺取法来优化上面的算法。

实际上，尺取法就是把两重循环变成了一个循环，在这个循环中一起处理i和j。复杂度也就从O(n2)变成了O(n)。仍以上面的反向扫描为例，代码是：

```
//用while实现：
int i = 0, j = n - 1;
while (i < j) {       //i和j在中间相遇。这样做还能防止i、j越界
        ......        //满足题意的操作
        i++;          //i从头扫到尾
        j--;          //j从尾扫到头
}
//用for实现：
for (int i = 0, j = n - 1; i < j; i++, j--) {
    ......
}
1234567891011
```

在尺取法中，这两个指针i、j，有两种扫描方向：

（a）反向扫描。i、j方向相反，i从头到尾，j从尾到头，在中间相会。

（b）同向扫描。i、j方向相同，都从头到尾，可以让j跑在i前面。

在leetcode的一篇文章中 常用的双指针技巧 https://leetcode-cn.com/circle/article/GMopsy/，把同向扫描的i、j指针称为“快慢指针”，把反向扫描的i、j指针称为“左右指针”，更加形象。快慢指针在序列上产生了一个大小可变的“滑动窗口”，有灵活的应用，例如3.1的“寻找区间和”问题。

下文分别按双指针的反向扫描和同向扫描，给出一些经典例子。文中也列举了一些可在线提交的题目，供练习。

2 反向扫描

2.1 找指定和的整数对

这个问题是尺取法最经典，也最简单直接的应用。

■问题描述

输入n (n≤100,000)个整数，放在数组a[]中。找出其中的两个数，它们之和等于整数m(假定肯定有解)。题中所有整数都是int型。

样例输入：

21 4 5 6 13 65 32 9 23

28

样例输出：

5 23

说明：样例输入的第一行是数组a[]，第2行是m = 28。样例输出5和23，相加得28。

■题解

为了说明尺取法的优势，下面给出四种方法：

（1）用两重循环暴力搜，枚举所有的取数方法，复杂度O(n2)，超时。暴力法不需要排序。

（2）二分法。首先对数组从小到大排序，复杂度O(nlogn)；然后，从头到尾处理数组中的每个元素a[i]，在大于a[i]的数中二分查找是否存在一个等于 m - a[i]的数，复杂度也是O(nlogn)。两部分相加，总复杂度仍然是O(nlogn)。

（3）Hash。分配一个hash空间s，把n个数放进去。逐个检查a[]中的n个数，例如a[i]，检查m - a[i]在s中是否有值，如果有，那么存在一个答案。复杂度是O(n)。
hash方法很快，但是需要一个额外的、可能很大的hash空间。

（4）尺取法。这是标准解法。首先对数组从小到大排序；然后，设置两个变量i和j，分别指向头和尾，i初值是0，j初值是n-1，然后让i和j逐渐向中间移动，检查a[i]+a[j]，如果大于m，就让j减1，如果小于m，就让i加1，直至a[i]+a[j] = m。排序复杂度O(nlogn)，检查的复杂度O(n)，合起来总复杂度O(nlogn)。

尺取法代码如下，注意可能有多个答案：

```
void find_sum(int a[], int n, int m){
     sort(a, a + n);      //先排序，复杂度O(nlogn)
     int i = 0, j = n - 1;    //i指向头，j指向尾
     while (i < j){           //复杂度O(n)
            int sum = a[i] + a[j];
            if (sum > m)   j--;
            if (sum < m)   i++;
            if (sum == m){
                cout << a[i] << " " << a[j] << endl;  //打印一种情况
                i++;          //可能有多个答案，继续
            }
      }
}
12345678910111213
```

在这个题目中，尺取法不仅效率高，而且不需要额外的空间。

把题目的条件改变一下，可以变化为类似的问题，例如：判断一个数是否为两个数的平方和。

这个题目，其实也能用同向扫描来做。请读者思考。

2.2 判断回文串

给一个字符串，判断它是不是回文串。

例题：hdu 2029： http://acm.hdu.edu.cn/showproblem.php?pid=2029

■问题描述

“回文串”是一个正读和反读都一样的字符串，比如“level”或者“noon”就是回文串。写一个程序判断

读入的字符串是否是“回文”。如果是，输出“yes”，否则输出“no”。

■题解

题目很简单，不做分析。下面是尺取法代码。

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n;
    cin >> n;                          //n是测试用例个数
    while(n--){
        string s;  cin >> s;           //读一个字符串
        bool ans = true;
        int i = 0, j = s.size() - 1;   //双指针
        while(i < j){
            if(s[i] != s[j]){
                ans = false;
                break;
            }
            i++;   j--;                //移动双指针
        }
        if(ans)   cout << "yes" << endl;
        else      cout << "no"  << endl;
    }
    return 0;
}
12345678910111213141516171819202l
```

稍微改变一下，类似的题目有：

（1）不区分大小写，忽略非英文字母，判断是否回文串。

提交地址：http://www.lintcode.com/problem/valid-palindrome/

这一题很简单。

（2）允许删除（或插入，本题只考虑删除）最多1个字符，判断是否能构成回文字符串。

提交地址：http://www.lintcode.com/problem/valid-palindrome-ii/

设反向扫描双指针为i、j。如果 s[i]和s[j]相同，i++、j-；如果s[i]和s[j]不同，那么，或者删除s[i]，或者删除s[j]，看剩下的字符串是否是回文串即可。

3 同向扫描

3.1 寻找区间和

这是用尺取法产生“滑动窗口”的典型例子。

■问题描述

给定一个长度为n的数组a[]和一个数s，在这个数组中找一个区间，使得这个区间之和等于s。输出区间的起点和终点位置。

样例输入：

15

6 1 2 3 4 6 4 2 8 9 10 11 12 13 14

6

样例输出：

0 0

1 3

5 5

6 7

说明：样例输入的第1行是n=15，第2行是数组a[]，第3行是区间和s=6。样例输出，共有4个情况。

■题解

指针i和j，i<=j，都从头向尾扫描，判断区间[i,j]的和是否等于s。

如何寻找区间和等于s的区间？如果简单地对i和j做二重循环，复杂度是O(n2)。用尺取法，复杂度O(n)，操作步骤是：

（1）初始值i=0、j=0，即开始都指向第一个元素a[0]。定义sum是区间[i, j]的和，初始值sum = a[0]。

（2）如果sum等于s，输出一个解。继续，把sum减掉元素a[i]，并把i往后移动一位。

（3）如果sum大于s，让sum减掉元素a[i]，并把i往后移动一位。

（4）如果sum小于s，把j往后挪一位，并把sum的值加上这个新元素。

在上面的步骤中，有2个关键技巧：

（1）滑动窗口的实现。窗口就是区间[i,j]，随着i和j从头到尾移动，窗口就“滑动”扫描了整个序列，检索了所有的数据。i和j并不是同步增加的，窗口像一只蚯蚓伸缩前进，它的长度是变化的，这个变化，正对应了对区间和的计算。

（2）sum的使用。如何计算区间和？暴力的方法是从a[i]到a[j]累加，但是，这个累加的复杂度是O(n)的，会超时。如果利用sum，每次移动i或j的时候，只需要把sum加或减一次，就得到了区间和，复杂度是O(1)。这是“前缀和”递推思想的应用。

下面是代码。

```
void findsum(int *a, int n, int s){
    int i = 0, j = 0;
    int sum = a[0];
    while(j < n){   //下面代码中保证 i<=j
        if(sum >= s){
            if(sum == s) printf("%d %d\n", i, j);
            sum -= a[i];
            i++;
            if(i>j) {sum = a[i]; j++;}  //防止i超过j
        }
        if(sum < s){
            j++;
            sum += a[j];
        }
    }
}
12345678910111213141516
```

“滑动窗口”的例子还有：

（1）给定一个序列，以及一个整数M；在序列中找M个连续递增的元素，使它们的区间和最大。

（2）给定一个序列，以及一个整数K；求一个最短的连续子序列，其中包含至少K个不同的元素。

在“4 典型题目”中有相似的题目。

3.2 数组去重

数组去重是很常见的操作，方法也很多，尺取法是其中优秀的算法。

■问题描述

给定数组a[]，长度为n，把数组中重复的数去掉。

■题解

下面给出两种解法：hash和尺取法。

hash。hash函数的特点是有冲突，利用这个特点去重。把所有的数插到 hash表里，用冲突过滤重复的数，就能得到不同的数。缺点是会耗费额外的空间。

尺取法。步骤是：

（1）将数组排序，这样那些重复的整数就会挤在一起。

（2）定义双指针i、j，初始值都指向a[0]。i和j都从头到尾扫描数组a[]。i指针走得快，逐个遍历整个数组；j指针走得慢，它始终指向当前不重复部分的最后一个数。也就是说，j用于获得不重复的数。

（3）扫描数组。快指针i++，如果此时a[i]不等于慢指针j指向的a[j]，就把j++，并且把a[i]复制到慢指针j的当前位置a[j]。

（4）i扫描结束后，a[0]到a[j]就是不重复数组。

4 典型题目

4.1 尺取法在链表中的应用

leetcode网站给出了尺取法在链表中的一些应用 常用的双指针技巧 https://leetcode-cn.com/circle/article/GMopsy/，下面列出4个。第1个给出说明，后面3个请直接看原文，有图解。

（1）判定链表中是否含有环

在单链表中，每个节点只知道下一个节点，所以一个指针无法判断链表中是否含有环。

如果链表中不含环，那么这个指针最终会遇到空指针 null，表示链表到头，可以判断该链表不含环。

但是如果链表中含有环，那么这个指针就会陷入死循环，因为环形数组中没有 null 指针作为尾部节点。

经典解法是用两个指针，一个跑得快，一个跑得慢。如果不含有环，跑得快的那个指针最终会遇到 null，说明链表不含环；如果含有环，快指针最终会超慢指针一圈，和慢指针相遇，说明链表含有环。下面是leetcode给出的java代码：

```
boolean hasCycle(ListNode head) {
    ListNode fast, slow;
    fast = slow = head;
    while (fast != null && fast.next != null) {
        fast = fast.next.next;          //快指针比慢指针快一倍
        slow = slow.next;
        if (fast == slow) return true; //快指针追上慢指针，说明有环
    }
    return false;
}
12345678910
```

（2）已知链表中含有环，返回这个环的起始位置

（3）寻找链表的中点

（4）寻找链表的倒数第 k 个元素

4.2 poj 3061

给定一个序列，包含N个正整数（10 <N <100 000），每个正整数均小于或等于10000，给定一个正整数S（S <100 000 000）。求序列中一个最短的连续区间，并且其区间和大于或等于S。

4.3 poj 2566

给定一个序列，包含n个整数（1<=n<=100000），以及一个整数t（0<=t<=1000000000）。求一段子序列，使它的区间和最接近t。输出该段子序列之和及左右端点。

4.4 hdu 5358

给定正整数序列a1,a2,...,an，定义S(i,j)是区间ai,ai+1,...,aj的和。计算:
∑ i = 1 n ∑ j = i n (∣ l o g 2 S (i , j) ∣ + 1) × (i + j) \sum\limits{{i=1}}^{{n}} \sum\limits{{j=i}}^{{n}} (|log_{2}S(i,j)|+1)\times(i+j)*i*=1∑*nj*=*i*∑*n*(∣*l**o**g*2*S*(*i*,*j*)∣+1)×(*i*+*j*)

4.5 洛谷p1102

A-B数对：给出一串数以及一个数字 C，要求计算出所有 A-B=C 的数对的个数。

4.6 uva 11572

给出 n个数，找尽量长的一个子序列，使得该子序列中没有重复的元素。
如果uva连不上，可以用https://vjudge.net/problem/UVA-11572，它能代理提交。

5 参考文献

[1]leetcode的尺取法题目
中文：https://leetcode-cn.com/tag/two-pointers/
英文：https://leetcode.com/articles/two-pointer-technique/
[2]leetcode的尺取法教程：
https://leetcode-cn.com/circle/article/GMopsy/

2020/11/12 wz摘自https://blog.csdn.net/weixin_43914593/article/details/104090474