

# 尺取法（双指针）

罗勇军 2020.1.26

本系列是这本算法教材的扩展资料：《算法竞赛入门到进阶》. 罗勇军、郭卫斌. 清华大学出版社

本文 web 地址：[https://blog.csdn.net/weixin\\_43914593/article/details/104090474](https://blog.csdn.net/weixin_43914593/article/details/104090474)

PDF 下载地址：<https://github.com/luoyongjun999/code> 其中的补充资料

如有建议，请联系：（1）QQ 群，567554289；（2）作者 QQ，15512356

## 目录

1 尺取法的概念.....	1
2 反向扫描.....	2
2.1 找指定和的整数对.....	2
2.2 判断回文串.....	3
3 同向扫描.....	4
3.1 寻找区间和.....	4
3.2 数组去重.....	5
4 典型题目.....	5
4.1 尺取法在链表中的应用.....	5
4.2 poj 3061.....	6
4.3 poj 2566.....	6
4.4 hdu 5358.....	6
4.5 洛谷 p1102.....	6
4.6 uva 11572.....	6
5 参考文献.....	6

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  $O(n^2)$ 。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
        .....
    }
```

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  $O(n^2)$  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--) {
    .....
}
```

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<sup>①</sup>，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  $n$  ( $n \leq 100,000$ ) 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  $\text{int}$  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  $O(n^2)$ ，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  $O(n \log n)$ ；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  $O(n \log n)$ 。两部分相加，总复杂度仍然是  $O(n \log n)$ 。

(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  $j$ ，

<sup>①</sup> 常用的双指针技巧 <https://leetcode-cn.com/circle/article/GMopsy/>

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  $O(n\log n)$ ，检查的复杂度  $O(n)$ ，合起来总复杂度  $O(n\log n)$ 。

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 - 1);    //先排序，复杂度  $O(n\log n)$ 
    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
        }
    }
}
```

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;
}

```

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  $j$ ， $i \leq j$ ，都从头到尾扫描，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  $O(n^2)$ 。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];
        }
    }
}
```

“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<sup>①</sup>，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;  
}
```

(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 kk 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 ( $1 \leq n \leq 100000$ )，以及一个整数 t ( $0 \leq t \leq 1000000000$ )。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  $a_1, a_2, \dots, a_n$ ，定义  $S(i, j)$  是区间  $a_i, a_{i+1}, \dots, a_j$  的和。计算

$$\sum_{i=1}^n \sum_{j=i}^n (|\log_2 S(i, j)| + 1) \times (i + j)$$

## 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<sup>②</sup>

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。

# 5 参考文献

<sup>①</sup> 常用的双指针技巧 <https://leetcode-cn.com/circle/article/GMopsy/>

<sup>②</sup> 如果 uva 连不上，可以用 <https://vjudge.net/problem/UVA-11572>，它能代理提交。

[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/>