1.两数之和

Unordered\_map

if(m.find(target-nums[i]) != m.end() && m[target-nums[i]]!=i)

                return {m[target-nums[i]],i};

2.两数之和（链表存数）

注意进位和链表长度不相同的情况

3.无重复字符的最长子串

用string来存放没有重复的字符串，一旦新加的字符在字符串中出现，删掉出现位置之前的字符串，并将新加字符串加载后面

5.最长回文子串

二维dp[len][len]，len=字符串长度，dp[i][j]表示的是下标为i和j之间(闭区间)的字符串是否为回文串，取值为1表示为真

用start和maxlen记录最长长度情况下的开始下标和长度

先初始化字符串长度为1和2的情况，长度为2的时候

if(i<len-1&&s[i]==s[i+1])

        {

            dp[i][i+1]=1;

            maxlen = 2;

            start = i;

        }

长度为3以上的时候，二重循环，外层为长度，内层为起始下标

for(int i=3;i<=len;i++)///从长度为3开始

    {

        for(int j=0;j<len-i+1;j++)

        {

            int k = j+i-1;

            if(s[j]==s[k] && dp[j+1][k-1]==1)

            {

                dp[j][k]=1;

                maxlen = i;

                start = j;

            }

        }

    }

6.Z型变换

用dx和dy两个取值为+-1表示方向，然后根据这个遍历

7.整数反转

(1)直接转字符串，问题是溢出问题和负数,还有前导0

(2)除10循环，考虑溢出

 while(x)

{

       int num = x % 10;

       x/=10;

       if(sum > INT\_MAX/10 || sum < INT\_MIN/10) return 0;//头文件limits.h

       sum = sum\*10 + num;

  }

8.字符串转整数

剥离有正负号的情况，flag存放符号

//INT\_MAX = 2147483647

while(str[i]>='0' && str[i]<='9')

        {

            int temp = str[i]-'0';

            if(flag == 1 && (sum>INT\_MAX/10 || (sum == INT\_MAX/10 && temp>7))) return INT\_MAX;

            else if(sum<INT\_MIN/10 || (sum == INT\_MIN/10 && temp>8)) return INT\_MIN;

            sum = sum\*10 + temp\*flag;

            i++;

        }

9.回文数

(1) if(x < 0 || (x % 10 == 0 && x != 0)) return false;

int revertedNumber = 0;

while(x > revertedNumber) {

revertedNumber = revertedNumber \* 10 + x % 10;

x /= 10;

}

return x == revertedNumber || x == revertedNumber/10;

(2)字符串

11.盛最多水的容器

双指针法，每次看左右哪边更低，就向中间靠近

#### 13.[罗马数字转整数](https://leetcode-cn.com/problems/roman-to-integer/)

左减右加

14.最长公共前缀

最简单的就是水平纵向遍历，先找最小长度的字符串，用其做基准

可以二分，分治

15.三数之和

双指针法

a+b+c=num?可以转换成b+c=num-a;

a做最左边下标，bc做双指针

16.最接近的三数之和

先排序，然后参考三数之和

17.电话号码的字母组合

回溯，dfs

18.四数之和

在三数之和基础上另外加了一重循环

19.删除链表倒数第k个结点

两次遍历

一次遍历（一次遍历使用快慢指针，快指针和慢指针距离为k，然后当快指针为nullptr时删除快指针后面那一个结点），注意链表长度等于k的情况，加一个哑结点解决

20.有效的括号

压栈，与栈顶比较

21.合并两个有序链表

两个指针比较遍历

22.括号生成

Dfs回溯，注意已经放置的左括号数一定≥右括号数

Dfs(v,s,n,n);

Dfs()

{

if(left==0 && right==0)

        {

            v.push\_back(s);

            return ;

        }

        else if(left >= 0&& right>=0)

        {

            if(left > right) return ;

            dfs(v,s+"(",left-1,right);

            dfs(v,s+")",left,right-1);

        }

}

23.合并k个已排序序列

基础是合并2个链表

然后用分别合并k中的链表

int len = lists.size();

        int interval=1;

        while (len > interval)

        {

            for (int i = 0; i < len - interval; i += 2\*interval)

            {

                lists[i] = mergeTwoLists(lists[i], lists[i + interval]);

            }

            interval\*=2;

        }

        return lists[0];

24.两两交换链表中的结点

新增一个哑结点，记得画图

a→b→c→d a→c→b→d;

26.删除排序树组中的重复项

快慢指针，当快慢指针值不一样时慢指针移动

27.移除元素

和26差不多，双指针，当快指针不是val时，慢指针移动

28.找子串

移动窗口

29.两数相除

考虑负数/-1溢出

将负数都转换为正数来计算

int div(long a, long b){  // 似乎精髓和难点就在于下面这几句

        if(a<b) return 0;

        long count = 1;

        long tb = b; // 在后面的代码中不更新b

        while((tb+tb)<=a){

            count = count + count; // 最小解翻倍

            tb = tb+tb; // 当前测试的值也翻倍

        }

        return count + div(a-tb,b);

    }

31.下一个排列

从右向左找到第一个nums[pos] > nums[pos-1]的地方，然后记录pos-1位置，翻转pos以后的元素位置，然后从pos向右找第一个大于他的位置tmp，交换nums[tmp]和nums[pos-1]

int pos = nums.size() - 1;

        while(pos > 0 && nums[pos] <= nums[pos-1]) pos--;

        reverse(nums.begin()+pos,nums.end());

        if(pos > 0)

        {

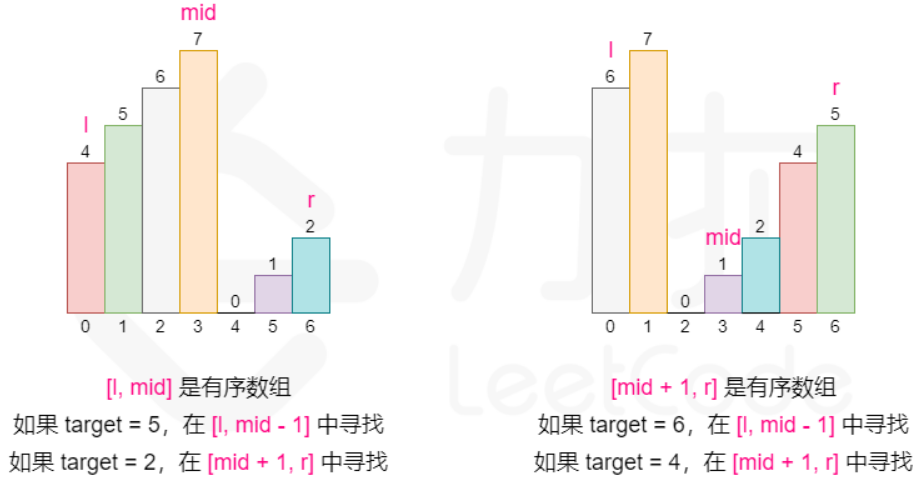
            auto iter = upper\_bound(nums.begin()+pos,nums.end(),nums[pos-1]);

            swap(nums[pos-1],\*iter);

        }

33.搜索旋转排序数组

变种二分，记得讨论情况



34. [在排序数组中查找元素的第一个和最后一个位置](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/)

两次二分，第一次找右边界，提前判断有没有，然后第二次找左边界

while(left<right)

        {

            int mid = (left + right)/2;

            if(nums[mid]>=target) right = mid;//第二次没等号

            else left = mid+1;

        }

35.搜索插入位置

就是二分，注意边界段错误

39.组合总数

Dfs剪枝

42.接雨水

两遍遍历

第一次从左往右

第二次从右往左

然后减去最大矩形和每个柱子的面积

46.全排列

Dfs回溯

50.pow

快速幂乘

53.最大子序和

一维Dp

55.跳跃游戏

Dp，用k记录能跳到的最大位置。

56.合并区间

公交车上下车问题，或者教室活动分配问题

双栈法即可

62.不同路径

二维dp[len+1][len+1]，初始化dp[1][1]=1;

dp[i][j] += dp[i-1][j] + dp[i][j-1];

63.不同路径2

一样的，不过加了障碍物，遇到dp[i][j]=0;

64.最小路径和

二维grid[i][j] += min(grid[i][j-1],grid[i-1][j]);

69.x的平方根

二分

if(x/mid >= mid && x/(mid+1) < (mid+1)) return mid;

70.爬楼梯

v[i] = v[i-1] + v[i-2];

98.验证二叉搜索树

中根遍历序列为升序

101.镜像/对称二叉树

bool isMirror(TreeNode \*left,TreeNode \*right)

    {

        if(left == nullptr && right == nullptr) return true;

        else if(left==nullptr || right==nullptr) return false;

        return ((left->val == right->val) && isMirror(left->left, right->right) && isMirror(left->right,right->left));

    }

102.二叉树层序遍历

Bfs+队列，简单点

Dfs也行

103.二叉树的锯齿状层序遍历

同样是bfs+队列，但是引入一个flag判断是正着读还是倒着读

104.二叉树最大深度

int maxDepth(TreeNode\* root) {

        if(root == nullptr) return 0;

        int left = maxDepth(root->left);

        int right = maxDepth(root->right);

        return (max(left,right)+1);

    }

136.只出现一次的数字

全部异或

141.环形链表

快慢指针即可

142.环形链表2，要找到环的入口

也是快慢指针，但是用弗洛伊德算法

if (head == nullptr || head->next == nullptr) {

            return nullptr;

        }

        ListNode\* slow = head;

        ListNode\* fast = head;

        do{

            if (fast == nullptr || fast->next == nullptr) {

                return nullptr;

            }

            slow = slow->next;

            fast = fast->next->next;

        }while (slow != fast);

        //找到slow=fast点为环中一个点

        slow = head;

        while(slow!=fast)

        {

            slow = slow->next;

            fast = fast->next;

        }

        return fast;

当slow=fast确定有环以后，slow回到开头，这时slow和fast一次都是走一步，相遇时就是环的入口

237.删除链表中的一个结点，时间复杂度要求O(1)

考虑两种情况：一是不是尾结点，那么将后一个的值赋值给本结点，然后将自己的next指向next的next

而是尾结点，直接判断后面是不是nullptr，是就直接赋值为nullptr