## [剑指 Offer 21. 调整数组顺序使奇数位于偶数前面](https://leetcode-cn.com/problems/diao-zheng-shu-zu-shun-xu-shi-qi-shu-wei-yu-ou-shu-qian-mian-lcof/)

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。

示例：

输入：nums = [1,2,3,4]

输出：[1,3,2,4]

注：[3,1,2,4] 也是正确的答案之一。

提示：

0 <= nums.length <= 50000

1 <= nums[i] <= 10000

**解：**

双向队列：

对于奇数，将其插入队列头部；对于偶数，将其插入队列尾部。

时间O(n) 空间O(n)

class Solution {

public:

    vector<int> exchange(vector<int>& nums) {

        deque<int> res;

        for(int i : nums){

            if(i & 1){

                res.push\_front(i);

            }

            else{

                res.push\_back(i);

            }

        }

        vector<int> a;

        for(int i : res){

            a.push\_back(i);

        }

        return a;

    }

};

双vector：

将奇数偶数分别存入两个vector，最后合并。

时间O(n) 空间O(n)

class Solution {

public:

    vector<int> exchange(vector<int>& nums) {

        vector<int> odd;

        vector<int> even;

        for(int i : nums){

            if(i & 1){

                odd.push\_back(i);

            }

            else{

                even.push\_back(i);

            }

        }

        odd.insert(odd.end(),even.begin(),even.end());

        return odd;

    }

};

双指针：

双指针分别从头尾进行遍历，若前指针为偶数而后指针为奇数则进行交换。

时间O(n) 空间O(1)

class Solution {

public:

    void swap(int &a, int &b){

        int tmp = a;

        a = b;

        b = tmp;

    }

    vector<int> exchange(vector<int>& nums) {

        if(nums.empty()){

            return {};

        }

        auto front\_iter = nums.begin();

        auto back\_iter = nums.end()-1;

        while(front\_iter < back\_iter){

            if(\*front\_iter & 1){

                front\_iter++;

                continue;

            }

            if(!(\*back\_iter & 1)){

                back\_iter--;

                continue;

            }

            swap(\*front\_iter,\*back\_iter);

            front\_iter++;

            back\_iter--;

        }

        return nums;

    }

};

改写sort函数：

将sort函数的比较规则改为奇数大于偶数。

时间O(nlogn) 空间O(1)

class Solution {

public:

    static bool cmp(int a, int b){

        return (a%2)>(b%2);

    }

    vector<int> exchange(vector<int>& nums) {

        if(nums.empty()){

            return {};

        }

        sort(nums.begin(),nums.end(),cmp);

        return nums;

    }

};

## [剑指 Offer 22. 链表中倒数第k个节点](https://leetcode-cn.com/problems/lian-biao-zhong-dao-shu-di-kge-jie-dian-lcof/)

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

示例：

给定一个链表: 1->2->3->4->5, 和 k = 2.

返回链表 4->5.

**解：**

双指针：

p指向头结点，q指向p后k-1个节点。遍历链表，p、q同时移动，当q指向链表尾时，返回p节点。

时间O(n) 空间O(1)

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* get\_q(ListNode\* p, int n){

        ListNode\* q = p;

        for(int i = 1; i < n; ++i){

            q = q->next;

        }

        return q;

    }

    ListNode\* getKthFromEnd(ListNode\* head, int k) {

        ListNode\* p = head;

        ListNode\* q = get\_q(p,k);

        while(q->next != NULL){

            p = p->next;

            q = q->next;

        }

        return p;

    }

};

辅助栈：

用栈存入所有节点，弹出k个。

时间O(n) 空间O(n)

## [剑指 Offer 24. 反转链表](https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/)

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

示例:

输入: 1->2->3->4->5->NULL

输出: 5->4->3->2->1->NULL

限制：

0 <= 节点个数 <= 5000

**解：**

双指针：

定义两个指针： pre 和 cur ；pre 在前 cur 在后。

每次让 pre 的 next 指向 cur ，实现一次局部反转。

局部反转完成之后， pre 和 cur 同时往前移动一个位置。

循环上述过程，直至 pre 到达链表尾部。

时间O(n) 空间O(1)

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* cur = NULL;

        ListNode\* pre = head;

        while (pre != NULL) {

            ListNode\* t = pre->next;

            pre->next = cur;

            cur = pre;

            pre = t;

        }

        return cur;

    }

};

## [剑指 Offer 25. 合并两个排序的链表](https://leetcode-cn.com/problems/he-bing-liang-ge-pai-xu-de-lian-biao-lcof/)

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

示例1：

输入：1->2->4, 1->3->4

输出：1->1->2->3->4->4

限制：

0 <= 链表长度 <= 1000

**解：**

双指针迭代：

p、q分别指向双链表头结点，建立一个新链表存储结果。比较当前值，小的插入结果链表中。若一链表遍历结束，直接将另一链表接入。

时间O(m+n) 空间O(m+n)

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

 \*     ListNode(int x) : val(x), next(NULL) {}

 \* };

 \*/

class Solution {

public:

    ListNode\* mergeTwoLists(ListNode\* l1, ListNode\* l2) {

        ListNode\* p = l1;

        ListNode\* q = l2;

        ListNode\* head = new ListNode(0);

        ListNode\* i = head;

        while(p != NULL && q != NULL){

            if(p->val < q->val){

                i->next = p;

                p = p->next;

            }

            else{

                i->next = q;

                q = q->next;

            }

            i = i->next;

        }

        i->next = q == NULL ? p : q;

        return head->next;

    }

};

## [剑指 Offer 26. 树的子结构](https://leetcode-cn.com/problems/shu-de-zi-jie-gou-lcof/)

输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

例如:

给定的树 A:

     3

    / \

   4   5

  / \

 1   2

给定的树 B：

   4

  /

 1

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

示例 1：

输入：A = [1,2,3], B = [3,1]

输出：false

示例 2：

输入：A = [3,4,5,1,2], B = [4,1]

输出：true

限制：

0 <= 节点个数 <= 10000

**解：**

递归：

边界判断：若两个树有一个为空返回false。

调用helper函数或对A树左子树或右子树继续递归。

helper()：递归基：B节点为空返回true。当A节点为空或AB节点val不一致时返回false。对于左子树和右子树进行递归。

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

 \* };

 \*/

class Solution {

public:

    bool isSubStructure(TreeNode\* A, TreeNode\* B) {

        if(A == NULL || B == NULL)

            return false;

        return helper(A,B) || isSubStructure(A->left,B) || isSubStructure(A->right,B);

    }

    bool helper(TreeNode\* A, TreeNode\* B){

        if(B == NULL) return true;

        if(A == NULL || A->val != B->val) return false;

        return helper(A->left,B->left) && helper(A->right,B->right);

    }

};

## [剑指 Offer 27. 二叉树的镜像](https://leetcode-cn.com/problems/er-cha-shu-de-jing-xiang-lcof/)

请完成一个函数，输入一个二叉树，该函数输出它的镜像。

例如输入：

     4

   /   \

  2     7

 / \   / \

1  3 6   9

镜像输出：

     4

   /   \

  7     2

 / \   / \

9   6 3   1

示例 1：

输入：root = [4,2,7,1,3,6,9]

输出：[4,7,2,9,6,3,1]

限制：

0 <= 节点个数 <= 1000

**解：**

层次遍历：将每个节点的左右子树互换。

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

 \* };

 \*/

class Solution {

public:

    TreeNode\* mirrorTree(TreeNode\* root) {

        if(!root){

            return NULL;

        }

        queue<TreeNode\*> s;

        s.push(root);

        while(!s.empty()){

            TreeNode\* node = s.front();

            s.pop();

            if(node->left) s.push(node->left);

            if(node->right) s.push(node->right);

            TreeNode\* tmp = node->left;

            node->left = node->right;

            node->right = tmp;

        }

        return root;

    }

};

## [剑指 Offer 28. 对称的二叉树](https://leetcode-cn.com/problems/dui-cheng-de-er-cha-shu-lcof/)

请实现一个函数，用来判断一棵二叉树是不是对称的。如果一棵二叉树和它的镜像一样，那么它是对称的。

例如，二叉树 [1,2,2,3,4,4,3] 是对称的。

    1

   / \

  2   2

 / \ / \

3  4 4  3

但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的:

    1

   / \

  2   2

   \   \

   3    3

示例 1：

输入：root = [1,2,2,3,4,4,3]

输出：true

示例 2：

输入：root = [1,2,2,null,3,null,3]

输出：false

限制：

0 <= 节点个数 <= 1000

**解：**

利用辅助队列一层层遍历，调整进队的顺序，相对应的节点两两入队，判断是否相等，再两两出队。

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

 \* };

 \*/

class Solution {

public:

    bool isSymmetric(TreeNode\* root) {

        if(!root) return true;

        queue<TreeNode\*> q;

        q.push(root->left);

        q.push(root->right);

        while(!q.empty()){

            TreeNode\* leftNode = q.front();

            q.pop();

            TreeNode\* rightNode = q.front();

            q.pop();

            if(!leftNode && !rightNode){

                continue;

            }

            if(!leftNode || !rightNode || (leftNode->val != rightNode->val)){

                return false;

            }

            q.push(leftNode->left);

            q.push(rightNode->right);

            q.push(leftNode->right);

            q.push(rightNode->left);

        }

        return true;

    }

};

## [剑指 Offer 29. 顺时针打印矩阵](https://leetcode-cn.com/problems/shun-shi-zhen-da-yin-ju-zhen-lcof/)

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

示例 1：

输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]

输出：[1,2,3,6,9,8,7,4,5]

示例 2：

输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

输出：[1,2,3,4,8,12,11,10,9,5,6,7]

限制：

0 <= matrix.length <= 100

0 <= matrix[i].length <= 100

**解：**

按层模拟：

从外圈到内圈一层层遍历，注意在从右到左，从下到上的过程中判断是否left<right && bottom<top。逐渐缩减范围，直到左右或上下界相交。

时间O(mn) 空间O(1)

class Solution {

public:

    vector<int> spiralOrder(vector<vector<int>>& matrix) {

        if(matrix.size()==0 || matrix[0].size()==0){

            return {};

        }

        vector<int> res;

        int left = 0,right = matrix[0].size()-,bottom = 0,top = matrix.size()-1;

        int i,j;

        while(left<=right && bottom<=top){

            for(i = bottom,j = left; j <= right; j++){

                res.push\_back(matrix[i][j]);

            }

            for(j = right,i = bottom + 1; i <= top; i++){

                res.push\_back(matrix[i][j]);

            }

            if(left<right && bottom<top)

            {for(i = top,j = right - 1; j >= left; j--){

                res.push\_back(matrix[i][j]);

            }

            for(j = left,i = top - 1; i >= bottom + 1; i--){

                res.push\_back(matrix[i][j]);

            }}

            left++;

            right--;

            bottom++;

            top--;

        }

        return res;

    }

};