剑指Offer刷题笔记

剑指 Offer 03. 数组中重复的数字

解法1：哈希表

```
class Solution {
public:
    int findRepeatNumber(vector<int>& nums) {
        vector<int> count(100010,0);
        for(int i=0;i<nums.size();i++)
            if(count[nums[i]]++) return nums[i];
        return -1;
    }
};
```

解法2：set容器

```
class Solution {
public:
    int findRepeatNumber(vector<int>& nums) {
        unordered_set<int> Set;
        for(int i=0;i<nums.size();i++){
            if(Set.find(nums[i])!=Set.end()) return nums[i];
            else Set.insert(nums[i]);
        }
        return -1;
    }
};
```

笔记：常见的三种哈希结构（数组，set，map）

数组略。

在C++中，set 和 map 分别提供以下三种数据结构，其底层实现以及优劣如下表所示：

集合	底层实现	是否有序	数值是否可以重复	能够更改数值	查询效率	增删效率

集合	底层实现	是否有序	数值是否可以重复	能够更改数值	查询效率	增删效率
std::set	红黑树	有序	否	否	O(log n)	O(log n)
std::multiset	红黑树	有序	是	是	O(log n)	O(log n)
std::unordered_set	红黑树	无序	否	否	O(1)	O(1)

std::unordered_set底层实现为哈希表，std::set 和std::multiset 的底层实现是红黑树，红黑树是一种平衡二叉搜索树，所以key值是有序的，但key不可以修改，改动key值会导致整棵树的错乱，所以只能删除和增加。

集合	底层实现	是否有序	数值是否可以重复	能够更改数值	查询效率	增删效率
std::map	红黑树	key有序	key不可重复	key不可修改	O(logn)	O(logn)
std::multimap	红黑树	key有序	key可重复	key不可修改	O(log n)	O(log n)
std::unordered_map	哈希表	key无序	key不可重复	key不可修改	O(1)	O(1)

std::unordered_map 底层实现为哈希表，std::map 和std::multimap 的底层实现是红黑树。同理，std::map 和std::multimap的key也是有序的（这个问题也经常作为面试题，考察对语言容器底层的理解）。

当我们要使用集合来解决哈希问题的时候，优先使用unordered_set，因为它的查询和增删效率是最优的，如果需要集合是有序的，那么就用set，如果要求不仅有序还要有重复数据的话，那么就用multiset。

那么再来看一下map ，在map 是一个key value 的数据结构，map中，对key是有限制，对value没有限制的，因为key的存储方式使用红黑树实现的。

其他语言例如：java里的HashMap ， TreeMap 都是一样的原理。可以灵活贯通。

set容器和map容器的用法

```
set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列
    size()
    empty()
    clear()
    begin()/end()
    ++, -- 返回前驱和后继，时间复杂度 O(logn)

    set/multiset
        insert()  插入一个数
        find()  查找一个数
        count()  返回某一个数的个数
        erase()
            (1) 输入是一个数x，删除所有x   O(k + logn)
            (2) 输入一个迭代器，删除这个迭代器
        lower_bound()/upper_bound()
            lower_bound(x)  返回大于等于x的最小的数的迭代器
            upper_bound(x)  返回大于x的最小的数的迭代器
    map/multimap
        insert()  插入的数是一个pair
        erase()  输入的参数是pair或者迭代器
        find()
```

```
        []  注意multimap不支持此操作。 时间复杂度是 O(logn)
        lower_bound()/upper_bound()

unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
    count()
    和上面类似，增删改查的时间复杂度是 O(1)
    不支持 lower_bound()/upper_bound()， 迭代器的++，--
```

set的使用举例（另外两种容器类似）

```
#include <iostream>
#include <set>
using namespace std;
int main() {
    set<int> s; // 定义一个空集合s
    s.insert(1); // 向集合s里面插入一个1
    cout << *(s.begin()) << endl; // 输出集合s的第一个元素 (前面的星号表示要对指针取值)
    for (int i = 0; i < 6; i++) {
        s.insert(i); // 向集合s里面插入i
    }
    for (auto it = s.begin(); it != s.end(); it++) { // 用迭代器遍历集合s里面的每一个元素
        cout << *it << " ";
    }
    cout << endl << (s.find(2) != s.end()) << endl; // 查找集合s中的值，如果结果等于s.end()表示未找到
(因为s.end()表示s的最后一个元素的下一个元素所在的位置)
    cout << (s.find(10) != s.end()) << endl; // s.find(10) != s.end()表示能找到10这个元素
    s.erase(1); // 删除集合s中的1这个元素
    cout << (s.find(1) != s.end()) << endl; // 这时候元素1就应该找不到啦～
    return 0;
}
```

map的使用举例（另外两种容器类似）

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
int main() {
    map<string, int> m; // 定义一个空的map m，键是string类型的，值是int类型的
    m["hello"] = 2; // 将key为"hello", value为2的键值对(key-value)存入map中
    cout << m["hello"] << endl; // 访问map中key为"hello"的value, 如果key不存在，则返回0
    cout << m["world"] << endl;
    m["world"] = 3; // 将"world"键对应的值修改为3
    m[","] = 1; // 设立一组键值对，键为"," 值为1
    // 用迭代器遍历，输出map中所有的元素，键用it->first获取，值用it->second获取
    for (auto it = m.begin(); it != m.end(); it++) {
        cout << it->first << " " << it->second << endl;
    }
    // 访问map的第一个元素，输出它的键和值
    cout << m.begin()->first << " " << m.begin()->second << endl;
    // 访问map的最后一个元素，输出它的键和值
    cout << m.rbegin()->first << " " << m.rbegin()->second << endl;
    // 输出map的元素个数
```

```
    cout << m.size() << endl;
    return 0;
}
```

剑指 Offer 04. 二维数组中的查找

解法1：二分查找

```
class Solution {
public:
    bool findNumberIn2DArray(vector<vector<int>>& matrix, int target) {
        for (const auto& row: matrix) {
            auto it = lower_bound(row.begin(), row.end(), target);
            if (it != row.end() && *it == target) {
                return true;
            }
        }
        return false;
    }
};
```

解法2：Z字形查找

旋转45°，类似二叉搜索树的找法，为最优解

```
class Solution {
public:
    bool findNumberIn2DArray(vector<vector<int>>& matrix, int target) {
        if(matrix.size()==0||matrix[0].size()==0) return false;
        int i=0,j=matrix[0].size()-1;
        while(i<matrix.size()&&j>=0){
            if(matrix[i][j]==target) return true;
            else if(matrix[i][j]<target) i++;
            else j--;
        }
        return false;
    }
};
```

笔记：vector容器的遍历方法

参考：https://blog.csdn.net/HW140701/article/details/78833486?ops_request_misc=&request_id=&biz_id=102&utm_term=vector%20%E9%81%8D%E5%8E%86&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-78833486.142^{v73}wechat,201^{v4}add_ask,239^{v2}insert_chatgpt&spm=1018.2226.3001.4187

```
#include <vector>
#include <iostream>

using namespace std;

int main()
{
    vector<int> vec(6, 6);
    //第一种遍历方式，下标
    for (int i = 0; i < vec.size(); ++i) {
        cout << vec[i] << endl;
    }
    //第二种遍历方式，迭代器
    for (vector<int>::iterator iter = vec.begin(); iter != vec.end(); iter++) {
        cout << *iter << endl;
    }
    //第三种遍历方式，auto关键字
    for (auto iter = vec.begin(); iter != vec.end(); iter++) {
        cout << *iter << endl;
    }
    //第四种遍历方式，auto关键字的另一种方式
    for (auto i : vec) {
        cout << i << endl;
    }
    return 0;
}
```

笔记：整数二分查找模板

```
bool check(int x) {/* ... */} // 检查x是否满足某种性质

// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check()判断mid是否满足性质
        else l = mid + 1;
    }
    return l;
}
// 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用:
```

```
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

笔记：浮点数二分模板

```
bool check(double x) {/* ... */} // 检查x是否满足某种性质

double bsearch_3(double l, double r)
{
    const double eps = 1e-6;   // eps 表示精度，取决于题目对精度的要求
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

笔记：lower_bound()与upper_bound()

参考：https://blog.csdn.net/weixin_42051815/article/details/115873582

1. 在数组中使用lower_bound()和upper_bound()

   用法：`int i_lower = lower_bound(arr, arr + n, val) - arr;`

   作用：lower_bound()函数返回数组 arr 中**大于等于** val 的第一个元素的地址，若arr中的元素均小于 val 则返回尾后插入val位置地址。

   用法：`int i_upper = lower_bound(arr, arr + n, val) - arr;`

   作用：upper_bound()函数返回数组 arr 中**大于** val 的第一个元素的地址，若 arr 中的元素均小于等于 val 则返回尾后插入val位置地址。

2. STL中的lower_bound()和upper_bound()

   用法与数组中基本一样，但在STL中，返回值为迭代器。但在vector中，可以是迭代器也可以是位置。

   (1) vector动态数组

   ```
   int main()
   ```

```
{
    vector<int> vec{ 0,1,5,8,10,11};//有序数组
    vector<int>::iterator it1 = lower_bound(vec.begin(), vec.end(), 11);
    bool flag1 = it1 == vec.end() - 1;
    cout << flag1 << endl;//it1指向最后一个元素的迭代器
    unsigned i_lower = lower_bound(vec.begin(), vec.end(), 11) - vec.begin();
    cout << "i_lower:" << i_lower << endl;//位置5

    vector<int>::iterator it2 = upper_bound(vec.begin(), vec.end(), 11);
    bool flag2 = it2 == vec.end();
    cout << flag2 << endl;//it2为尾后迭代器
    unsigned j_upper = upper_bound(vec.begin(), vec.end(), 11) - vec.begin();
    cout << "j_upper:" << j_upper << endl;//位置6
    return 0;
}
```

运行结果:

```
1
i_lower:5
1
j_upper:6
```

(2) set集合

```
#include<algorithm>
#include<set>
using namespace std;

int main()
{
    set<int> s{ 0,2,5,8,11};//原本就有序
    set<int>::iterator it1 = s.lower_bound(2);
    cout << *it1 << endl;
    set<int>::iterator it2 = s.upper_bound(2);
    cout << *it2 << endl;
    return 0;
}
```

运行结果:

```
2
5
```

(3) map字典

```
#include<algorithm>
#include<map>
using namespace std;

int main()
{
    map<int, int> m{ {1,2},{2,2},{9,2},{7,2} };//有序
    map<int, int>::iterator it1 = m.lower_bound(2);
    cout << it1->first << endl;//it1->first=2
    map<int, int>::iterator it2 = m.upper_bound(2);
    cout << it2->first << endl;//it2->first=7;
    return 0;
}
```

运行结果：

```
2
7
```

剑指 Offer 05. 替换空格

解法1

```
class Solution {
public:
    string replaceSpace(string s) {
        string array;
        for(auto &c : s)
            if(c == ' ') array+="%20";
            else array.push_back(c);
        return array;
    }
};
```

剑指 Offer 06. 从尾到头打印链表

解法1

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
```

```
 */
class Solution {
public:
    vector<int> reversePrint(ListNode* head) {
        ListNode* node=head;
        vector<int> result;
        while(node!=NULL){
            result.push_back(node->val);
            node=node->next;
        }
        reverse(result.begin(),result.end());
        return result;
    }
};
```

笔记：reverse()

```
//翻转数组
//头文件
#include <algorithm>
//使用方法
reverse(a, a+n);//n为数组中的元素个数

//翻转字符串
//用法为
reverse(str.begin(), str.end());

//翻转向量
//用法
reverse(vec.begin(), vec.end());
```

剑指 Offer 07. 重建二叉树

解法1： 递归+分治

```
/**
 * 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:
```

```
    vector<int> pre_order,in_order;
    TreeNode* traversal(int l1,int r1,int l2,int r2)
    {
        if(l1>r1||l2>r2) return NULL;
        int mid;
        for(mid=l2;mid<=r2;mid++)
            if(in_order[mid]==pre_order[l1]) break;
        int len1=mid-l2;
        TreeNode *node=new TreeNode(pre_order[l1]);
        node->left=traversal(l1+1,l1+len1,l2,mid-1);
        node->right=traversal(l1+len1+1,r1,mid+1,r2);
        return node;
    }
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        pre_order=preorder,in_order=inorder;
        int l1=0,r1=preorder.size()-1,l2=0,r2=inorder.size()-1;
        TreeNode *root=traversal(l1,r1,l2,r2);
        return root;
    }
};
```

解法2：递归+分治（使用map容器优化）

```
/**
 * 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:
    vector<int> pre_order,in_order;
    map<int,int> index; //index为map容易，用于记录中序遍历位置
    TreeNode* traversal(int l1,int r1,int l2,int r2)
    {
        if(l1>r1||l2>r2) return NULL;
        int mid=index[pre_order[l1]]; //使用index直接得到mid的位置
        int len1=mid-l2;
        TreeNode *node=new TreeNode(pre_order[l1]);
        node->left=traversal(l1+1,l1+len1,l2,mid-1);
        node->right=traversal(l1+len1+1,r1,mid+1,r2);
        return node;
    }
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        pre_order=preorder,in_order=inorder;
        for(int i=0;i<inorder.size();i++) index[inorder[i]]=i;  //index记录中序遍历每个数的位置
        int l1=0,r1=preorder.size()-1,l2=0,r2=inorder.size()-1;
```

```
        TreeNode *root=traversal(l1,r1,l2,r2);
        return root;
    }
};
```

解法3：迭代（待理解）

```
/**
 * 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* buildTree(vector<int>& preorder, vector<int>& inorder) {
        if(preorder.size()==0) return NULL;
        TreeNode* root=new TreeNode(preorder[0]);
        stack<TreeNode*> st;
        int indexInorder=0;
        st.push(root);
        for(int i=1;i<preorder.size();i++){
            int preorderValue=preorder[i];
            TreeNode* node=st.top();
            if(node->val!=inorder[indexInorder]){
                node->left=new TreeNode(preorder[i]);
                st.push(node->left);
            }
            else{
                while(!st.empty()&&st.top()->val==inorder[indexInorder]){
                    node=st.top();
                    st.pop();
                    indexInorder++;
                }
                node->right=new TreeNode(preorderValue);
                st.push(node->right);
            }
        }
        return root;
    }
};
```

剑指 Offer 09. 用两个栈实现队列

解法1

```
class CQueue {
private:
    stack<int> s1,s2;  //s1用来出队,s2用来入队
public:
    CQueue() {
        while(!s1.empty()) s1.pop();
        while(!s2.empty()) s2.pop();
    }

    void appendTail(int value) {
        s2.push(value); //入队直接入到s2
    }

    int deleteHead() {
        if(s1.empty()&&s2.empty()) return -1;  //在两个栈都是空的情况下返回-1
        else if(s1.empty()){  //如果s1为空，则将s2中所有元素依次加入到s1中
            while(!s2.empty()){
                s1.push(s2.top());
                s2.pop();
            }
        }
        int x=s1.top();  //s1栈顶元素出栈，完场出队操作
        s1.pop();
        return x;
    }
};

/**
 * Your CQueue object will be instantiated and called as such:
 * CQueue* obj = new CQueue();
 * obj->appendTail(value);
 * int param_2 = obj->deleteHead();
 */
```

剑指 Offer 10- I. 斐波那契数列

解法1

```
class Solution {
public:
    int fib(int n) {
        int N=1e9+7;
        if(n<=1) return n;
        int a=0,b=1;
        for(int i=2;i<=n;i++){
            int temp=b;
            b=(a+b)%N;
```

```
            a=temp;
        }
        return b;
    }
};
```

剑指 Offer 10- II. 青蛙跳台阶问题

解法1

```
class Solution {
public:
    int numWays(int n) {
        int f[101],N=1e9+7;
        f[0]=1,f[1]=1;
        for(int i=2;i<=n;i++)
            f[i]=(f[i-2]+f[i-1])%N;
        return f[n];
    }
};
```

剑指 Offer 11. 旋转数组的最小数字

解法1：二分查找

```
class Solution {
public:
    int minArray(vector<int>& numbers) {
        int l=0,r=numbers.size()-1;
        while(l<numbers.size()&&numbers[l]==numbers[numbers.size()-1]) l++;
        if(l==numbers.size()) return numbers[0];
        while(l<r){
            int mid=l+(r-l)/2;
            if(numbers[mid]<=numbers[numbers.size()-1]) r=mid;
            else l=mid+1;
        }
        return numbers[l];
    }
};
```

剑指 Offer 12. 矩阵中的路径

解法1： DFS+剪枝

```
class Solution {
public:
    bool traversal(int x,int y,int index,vector<vector<char>>& board,string word,
vector<vector<bool>>& visit){
        if(index==word.length()-1&&board[x][y]==word[index]) return true;
        else if(board[x][y]!=word[index]) return false;
        int dx[]={0,1,0,-1};
        int dy[]={1,0,-1,0};
        for(int i=0;i<4;i++)
            if(x+dx[i]>=0&&x+dx[i]<board.size()&&y+dy[i]>=0&&y+dy[i]<board[0].size())
                if(visit[x+dx[i]][y+dy[i]]==false){
                    visit[x+dx[i]][y+dy[i]]=true;
                    if(traversal(x+dx[i],y+dy[i],index+1,board,word,visit)) return true;
                    visit[x+dx[i]][y+dy[i]]=false;
                }
        return false;
    }
    bool exist(vector<vector<char>>& board, string word) {
        vector<vector<bool>> visit(board.size(),vector<bool>(board[0].size(),false));
        for(int i=0;i<board.size();i++)
            for(int j=0;j<board[0].size();j++)
                if(board[i][j]==word[0]){
                    visit[i][j]=true;
                    if(traversal(i,j,0,board,word,visit)==true) return true;
                    visit[i][j]=false;
                }
        return false;
    }
};
```

剑指 Offer 14- I. 剪绳子

解法1

```
class Solution {
public:
    int cuttingRope(int n) {
        if(n==2) return 1;
        if(n==3) return 2;
        if(n==4) return 4;
        int result=1;
        while(n>4){
            result*=3;
            n-=3;
        }
        result*=n;
        return result;
```

```
    }
};
```

剑指 Offer 14- II. 剪绳子 II

解法1

```
class Solution {
public:
    int cuttingRope(int n) {
        int mod=1e9+7;
        if(n==2) return 1;
        if(n==3) return 2;
        if(n==4) return 4;
        int result=1;
        while(n>4){
            int temp=result;
            result=(result+temp)%mod;
            result=(result+temp)%mod;
            n-=3;
        }
        int temp=result;
        for(int i=0;i<n-1;i++) result=(result+temp)%mod;
        return result;
    }
};
```

剑指 Offer 15. 二进制中1的个数

解法1：位运算

```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int count=0;
        while(n){
            if(n&1) count++;
            n>>=1;
        }
        return count;
    }
};
```

剑指 Offer 16. 数值的整数次方

解法1：快速幂

```
class Solution {
public:
    double myPow(double x, int n) {
        double result=1;
        bool flag=true;
        long long k=n;
        if(k<0) flag=false,k=-(long long)n;
        while(k){
            if(k&1) result*=x;
            x*=x;
            k>>=1;
        }
        if(flag==false) result=1/result;
        return result;
    }
};
```

笔记：快速幂模板

```
// 求m^k mod p, 时间复杂度为 O(log k)
long long(int m,int k,int p){
    long long res=1&p,t=m;
    while(k){
        if(k&1) res=res*t%p;
        t=t*t;
        k>>=1;
    }
    return res;
}
```

剑指 Offer 17. 打印从1到最大的n位数

解法1

```
class Solution {
public:
    vector<int> printNumbers(int n) {
        vector<int> res;
        int count=0;
        for(int i=0;i<n;i++) count=count*10+9;
        for(int i=1;i<=count;i++) res.push_back(i);
        return res;
    }
};
```

剑指 Offer 18. 删除链表的节点

解法1

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteNode(ListNode* head, int val) {
        ListNode* dummyHead=new ListNode(0);  //添加链表虚拟头结点
        dummyHead->next=head;
        ListNode *pre=dummyHead,*p=head;
        while(p){
            if(p->val==val){
                pre->next=p->next;
                break;
            }
            pre=pre->next;
            p=p->next;
        }
        return dummyHead->next;
    }
};
```

剑指 Offer 19. 正则表达式匹配

解法1：动态规划（待记忆）

```
class Solution {
public:
    bool isMatch(string s, string p) {
        int m=s.size(),n=p.size();
        vector<vector<bool>> dp(m+1,vector<bool>(n+1,false));
        dp[0][0]=true;
        for(int i=0;i<=m;i++){
            for(int j=1;j<=n;j++){
                if(i>=1&&(s[i-1]==p[j-1]||p[j-1]=='.')) dp[i][j]=dp[i-1][j-1];
                else if(p[j-1]=='*'&&j>=2){
                    dp[i][j]=dp[i][j]||dp[i][j-2];
                    if(i>=1&&(s[i-1]==p[j-2]||p[j-2]=='.')) dp[i][j]=dp[i][j]||dp[i-1][j];
                }
            }
        }
        return dp[m][n];
    }
};
```

剑指 Offer 20. 表示数值的字符串

解法1：一步一步debug（待记忆）

```
class Solution {
public:
    bool isNumber(string s) {
        //先去掉s前后的空格
        string str;
        int i=0,j=s.length()-1;
        while(i<s.length()&&s[i]==' ') i++;
        while(j>=0&&s[j]==' ') j--;
        for(int k=i;k<=j;k++) str+=s[k];
        if(str.length()==0) return false;

        //若字符串开始为+/-号，后移一位
        int index=0;
        if(str[index]=='+'||str[index]=='-') index++;
        if(index==str.length()) return false;//字符串只有一个+/-号，结果为false

        //检查字符串直到遇见第一个./e/E符号退出
        while(index<str.length()&&str[index]!='.'&&str[index]!='e'&&str[index]!='E'){
            if(str[index]<'0'||str[index]>'9') return false;//中途出现空格或其他符号,结果为false
            else index++;
        }

        //根据第一个循环结果决定下一步操作
        if(index==str.length()) return true;//字符串表示整数，结果为true
```

```
        else if(str[index]=='.'){//遇见小数点.
            if(index>0&&str[index-1]>='0'&&str[index-1]<='9');//小数点前面至少一位数字
            else if(index+1==str.length()||str[index+1]<'0'||str[index+1]>'9') return false;//小数点前面没数字，后面也没数字（两种情况：后面没字符，或字符为空格或其他符号）
            index++;
        }

        //如果第一个环结束在e/E，则该循环不执行；如果第一个循环结束在小数点并跳一位后，则继续寻找e/E
        while(index<str.length()&&str[index]!='e'&&str[index]!='E'){
            if(str[index]<'0'||str[index]>'9') return false;//中途出现空格或其他符号,结果为false
            else index++;
        }

        //根据第二个循环结果决定下一步操作
        if(index==str.length()) return true;//字符串表示整数或小数（不带e/E）
        else{
            if(index>0&&(str[index-1]>='0'&&str[index-1]<='9'||str[index-1]=='.'));//e/E前面至少一位数字或小数点
            else if(index>=0||index==str.length()-1) return false;//e/E前面有不是数字或小数点的字符，或e/E后面连一位数字都没有
            index++;
        }

        if(str[index]=='+'||str[index]=='-') index++;//e/E后面是+/-号，后移一位
        if(index==str.length()) return false;//e/E后面只有一个+/-号，结果为false
        //第三个循环，确定e/E后面是整数，否则为false
        while(index<str.length()){
            if(str[index]<'0'||str[index]>'9') return false;
            else index++;
        }
        return true;
    }
};
```

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

解法1： 双指针

```
class Solution {
public:
    vector<int> exchange(vector<int>& nums) {
        int i=0,j=nums.size()-1;
        while(true){
            while(i<j&&nums[i]%2==1) i++;
            while(i<j&&nums[j]%2==0) j--;
            if(i<j) swap(nums[i],nums[j]);
            else break;
        }
        return nums;
    }
};
```

剑指 Offer 22. 链表中倒数第k个节点

解法1:快慢指针

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* getKthFromEnd(ListNode* head, int k) {
        ListNode *p=head,*q=head;
        while(k--&&q!=NULL) q=q->next;
        if(k>=0) return NULL;
        while(q!=NULL){
            p=p->next;
            q=q->next;
        }
        return p;
    }
};
```

剑指 Offer 24. 反转链表

解法1：头插法

```
/**
 * 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* dummyHead=new ListNode(0),*p;
        while(head){
            p=head;
            head=head->next;
            p->next=dummyHead->next;
            dummyHead->next=p;
        }
        return dummyHead->next;
    }
};
```

剑指 Offer 25. 合并两个排序的链表

解法1：归并

```
/**
 * 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 *dummyHead=new ListNode(0),*p=dummyHead;
        while(l1&&l2){
            if(l1->val<=l2->val){
                p->next=l1;
                l1=l1->next;
            }
            else{
                p->next=l2;
                l2=l2->next;
            }
```

```
            p=p->next;
        }
        if(l1) p->next=l1;
        else p->next=l2;
        return dummyHead->next;
    }
};
```

剑指 Offer 26. 树的子结构

解法1： 递归（背住）

```
/**
 * 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 isSub(TreeNode* A, TreeNode* B){
        if(B==NULL) return true;
        else if(A==NULL||A->val!=B->val) return false;
        return isSub(A->left,B->left)&&isSub(A->right,B->right);
    }
    bool isSubStructure(TreeNode* A, TreeNode* B) {
        if(A==NULL||B==NULL) return false;
        return isSub(A,B)||isSubStructure(A->left,B)||isSubStructure(A->right,B);
    }
};
```

剑指 Offer 27. 二叉树的镜像

解法1： 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:
    TreeNode* mirrorTree(TreeNode* root) {
        if(root==NULL) return NULL;
        TreeNode* p=root->left;
        root->left=mirrorTree(root->right);
        root->right=mirrorTree(p);
        return root;
    }
};
```

解法2：层序遍历

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        queue<TreeNode*> que;
        que.push(root);
        while(!que.empty())
        {
            TreeNode* node=que.front();
            que.pop();
            if(node==NULL) continue;
            swap(node->left,node->right);
            que.push(node->left);
            que.push(node->right);
        }
        return root;
    }
};
```

剑指 Offer 28. 对称的二叉树

解法1：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 isSym(TreeNode* leftNode,TreeNode* rightNode){
        if(leftNode==NULL&&rightNode==NULL) return true;
        else if(leftNode==NULL||rightNode==NULL) return false;
        else if(leftNode->val!=rightNode->val) return false;
        else return isSym(leftNode->left,rightNode->right)&&isSym(leftNode->right,rightNode->left);
    }
    bool isSymmetric(TreeNode* root) {
        if(root==NULL) return true;
        return isSym(root->left,root->right);
    }
};
```

解法2：层序遍历

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if(root==NULL) return true;
        queue<TreeNode*> que;
        que.push(root->left),que.push(root->right);
        while(!que.empty())
        {
            TreeNode *left_node=que.front();
            que.pop();
            TreeNode *right_node=que.front();
            que.pop();
```

```
            if(left_node==NULL&&right_node==NULL) continue;
            else if(left_node==NULL||right_node==NULL) return false;
            else if(left_node->val!=right_node->val) return false;
            que.push(left_node->left),que.push(right_node->right);
            que.push(left_node->right),que.push(right_node->left);
        }
        return true;
    }
};
```

剑指 Offer 29. 顺时针打印矩阵

解法1： 模拟

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        if(matrix.size()==0||matrix[0].size()==0) return {};
        int m=matrix.size(),n=matrix[0].size();
        vector<int> v;
        int u=0,d=m-1,l=0,r=n-1;
        while(u<=d&&l<=r)
        {
            for(int i=l;i<=r;i++) v.push_back(matrix[u][i]);
            if(++u>d) break;
            for(int i=u;i<=d;i++) v.push_back(matrix[i][r]);
            if(--r<l) break;
            for(int i=r;i>=l;i--) v.push_back(matrix[d][i]);
            if(--d<u) break;
            for(int i=d;i>=u;i--) v.push_back(matrix[i][l]);
            if(++l>r) break;
        }

        return v;
    }
};
```

剑指 Offer 30. 包含min函数的栈

解法1

```
class MinStack {
    stack<int> st,min_st;
public:
    /** initialize your data structure here. */
    MinStack() {
```

```
        while(!st.empty()) st.pop();
        while(!min_st.empty()) min_st.pop();
    }

    void push(int x) {
        st.push(x);
        if(min_st.empty()||x<=min_st.top()) min_st.push(x);
    }

    void pop() {
        if(!min_st.empty()&&st.top()==min_st.top()) min_st.pop();
        if(!st.empty()) st.pop();
    }

    int top() {
        if(!st.empty()) return st.top();
        return -1;
    }

    int min() {
        if(!min_st.empty()) return min_st.top();
        return -1;
    }
};

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack* obj = new MinStack();
 * obj->push(x);
 * obj->pop();
 * int param_3 = obj->top();
 * int param_4 = obj->min();
 */
```

剑指 Offer 31. 栈的压入、弹出序列

解法1

```
class Solution {
public:
    bool validateStackSequences(vector<int>& pushed, vector<int>& popped) {
        int i=0,j=0;
        stack<int> st;
        while(i<pushed.size()){
            if(i<pushed.size()&&pushed[i]!=popped[j]){
                if(!st.empty()&&st.top()==popped[j]) st.pop(),j++;
                else st.push(pushed[i]),i++;
            }
            else i++,j++;
        }
```

```
        while(!st.empty()){
            if(st.top()==popped[j]){
                st.pop();
                j++;
            }
            else return false;
        }
        return true;
    }
};
```

剑指 Offer 32 - I. 从上到下打印二叉树

解法1： 层序遍历

```
/**
 * 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:
    vector<int> levelOrder(TreeNode* root) {
        vector<int> res;
        queue<TreeNode*> que;
        if(root) que.push(root);
        while(!que.empty()){
            TreeNode* node=que.front();
            que.pop();
            res.push_back(node->val);
            if(node->left) que.push(node->left);
            if(node->right) que.push(node->right);
        }
        return res;
    }
};
```

剑指 Offer 32 - II. 从上到下打印二叉树 II

解法1： 层序遍历

```
/**
 * 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:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        queue<TreeNode*> que;
        if(root) que.push(root);
        while(!que.empty()){
            int size=que.size();
            vector<int> path;
            for(int i=0;i<size;i++){
                TreeNode* node=que.front();
                que.pop();
                path.push_back(node->val);
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
            res.push_back(path);
        }
        return res;
    }
};
```

剑指 Offer 32 - III. 从上到下打印二叉树 III

解法1： 层序遍历

```
/**
 * 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:
    vector<vector<int>> levelOrder(TreeNode* root) {
```

```
        vector<vector<int>> res;
        queue<TreeNode*> que;
        if(root) que.push(root);
        bool flag=false;
        while(!que.empty()){
            int size=que.size();
            vector<int> path;
            for(int i=0;i<size;i++){
                TreeNode* node=que.front();
                que.pop();
                path.push_back(node->val);
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
            if(flag) reverse(path.begin(),path.end());
            flag=!flag;
            res.push_back(path);
        }
        return res;
    }
};
```

剑指 Offer 33. 二叉搜索树的后序遍历序列

解法1： 递归+分治

```
class Solution {
public:
    bool verify(vector<int>& postorder,int left,int right){
        if(left>=right) return true;
        int pivot=postorder[right];
        int l1=left,r1,l2=left,r2=right-1;
        while(postorder[l2]<pivot) l2++;
        r1=l2-1;
        for(int i=l2;i<=r2;i++)
            if(postorder[i]<pivot) return false;
        return verify(postorder,l1,r1)&&verify(postorder,l2,r2);
    }
    bool verifyPostorder(vector<int>& postorder) {
        return verify(postorder,0,postorder.size()-1);
    }
};
```

剑指 Offer 34. 二叉树中和为某一值的路径

解法1： 回溯

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> res;
    vector<int> path;
    void traversal(TreeNode *node,int sum,int target){
        if(node->left==NULL&&node->right==NULL){
            if(sum==target) res.push_back(path);
            return;
        }
        if(node->left){
            path.push_back(node->left->val);
            sum+=node->left->val;
            traversal(node->left,sum,target);
            sum-=node->left->val;
            path.pop_back();
        }
        if(node->right){
            path.push_back(node->right->val);
            sum+=node->right->val;
            traversal(node->right,sum,target);
            sum-=node->right->val;
            path.pop_back();
        }
    }
    vector<vector<int>> pathSum(TreeNode* root, int target) {
        if(root){
            path.push_back(root->val);
            traversal(root,root->val,target);
            path.pop_back();
        }
        return res;
    }
};
```

剑指 Offer 35. 复杂链表的复制

解法1：回溯+哈希表（O(n)+O(n)）

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = NULL;
        random = NULL;
    }
};
*/
class Solution {
public:
    unordered_map<Node*,Node*> cachedNode;
    Node* copyRandomList(Node* head) {
        if(head==NULL) return NULL;
        else if(cachedNode.count(head)==NULL){
            Node* newHead=new Node(head->val);
            cachedNode[head]=newHead;
            newHead->next=copyRandomList(head->next);
            newHead->random=copyRandomList(head->random);
        }
        return cachedNode[head];
    }
};
```

解法2：迭代+节点拆分（O(n)+O(1)）

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = NULL;
        random = NULL;
    }
};
*/
class Solution {
```

```
public:
    Node* copyRandomList(Node* head) {
        if(head==NULL) return NULL;
        for(Node* node=head;node!=NULL;node=node->next->next){
            Node* newNode=new Node(node->val);
            newNode->next=node->next;
            node->next=newNode;
        }
        for(Node* node=head;node!=NULL;node=node->next->next){
            Node* newNode=node->next;
            if(node->random) newNode->random=node->random->next;
            else newNode->random=NULL;
        }
        Node* newHead=head->next;
        for(Node* node=head;node!=NULL;node=node->next){
            Node* newNode=node->next;
            node->next=newNode->next;
            if(newNode->next) newNode->next=newNode->next->next;
            else newNode->next=NULL;
        }
        return newHead;
    }
};
```

剑指 Offer 36. 二叉搜索树与双向链表

解法

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* left;
    Node* right;

    Node() {}

    Node(int _val) {
        val = _val;
        left = NULL;
        right = NULL;
    }

    Node(int _val, Node* _left, Node* _right) {
        val = _val;
        left = _left;
        right = _right;
    }
};
```

```
*/
class Solution {
public:
    Node *head=NULL,*pre=NULL;
    void traversal(Node* node){
        if(node==NULL) return;
        traversal(node->left);
        Node* newNode=new Node(node->val);
        if(pre==NULL){
            head=newNode;
            head->right=head;
            head->left=head;
        }
        else{
            pre->right->left=newNode;
            newNode->right=pre->right;
            pre->right=newNode;
            newNode->left=pre;
        }
        pre=newNode;
        traversal(node->right);
    }
    Node* treeToDoublyList(Node* root) {
        traversal(root);
        return head;
    }
};
```

剑指 Offer 37. 序列化二叉树

解法1（该解法会超时）

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Codec {
public:

    // Encodes a tree to a single string.
    string serialize(TreeNode* root) {
        string res="[";
        queue<TreeNode*> que;
        que.push(root);
        while(true){
```

```
            int flag=false;
            string temp;
            int size=que.size();
            for(int i=0;i<size;i++){
                TreeNode* node=que.front();
                que.pop();
                if(node!=root) temp+=",";
                if(node){
                    flag=true;
                    temp+=to_string(node->val);
                    que.push(node->left);
                    que.push(node->right);
                }
                else{
                    temp+="null";
                    que.push(NULL);
                    que.push(NULL);
                }
            }
            if(flag==true) res+=temp;
            else break;
        }
        res+="]";
        return res;
    }

    // Decodes your encoded data to tree.
    TreeNode* deserialize(string data) {
        if(data=="[]") return NULL;
        string temp;
        int count=1;
        TreeNode* head=NULL;
        queue<TreeNode*> que;
        for(int i=1;i<data.length();i++){
            if(data[i]!=','&&data[i]!=']') temp+=data[i];
            else{
                if(temp=="null") que.push(NULL);
                else{
                    TreeNode* node=new TreeNode(stoi(temp));
                    que.push(node);
                    if(head==NULL) head=node;
                    else{
                        TreeNode *pre=que.front();
                        if(count%2==0) pre->left=node;
                        else pre->right=node;
                    }
                }
                temp="";
                if(count!=1&&count%2) que.pop();
                count++;
            }
        }
        return head;
    }
```

```
};

// Your Codec object will be instantiated and called as such:
// Codec codec;
// codec.deserialize(codec.serialize(root));
```

剑指 Offer 39. 数组中出现次数超过一半的数字

解法1：哈希表（map）

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        map<int,int> hash;
        for(int i=0;i<nums.size();i++){
            hash[nums[i]]++;
            if(hash[nums[i]]>nums.size()/2) return nums[i];
        }
        return -1;
    }
};
```

解法2：遍历

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int result=nums[0],count=1;
        for(int i=1;i<nums.size();i++){
            if(count==0) result=nums[i];
            if(result==nums[i]) count++;
            else count--;
        }
        return result;
    }
};
```

剑指 Offer 40. 最小的k个数