剑指offer题解

前序

oj网站为牛客网

1.二维数组中的查找

题目描述

在一个二维数组中（每个一维数组的长度相同），每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

题解

```
class Solution {
public:
    bool Find(int target, vector<vector<int> > array) {
        bool find=false;

        if(array.size()>0&&array[0].size()>0){
            int row=0;
            int column=array[0].size()-1;
            while(row<array.size()&&column>=0){
                if(array[row][column]>target)
                    column--;
                else if(array[row][column]<target)
                    row++;
                else {
                    find= true;
                    break;
                }
            }
            return find;
        }
        return find;
    }
};
```

2.替换空格

题目描述

请实现一个函数，将一个字符串中的每个空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

题解

```
class Solution {
public:
    void replaceSpace(char *str,int length) {
        if(str==NULL||length<0)
            return;
        int oldlength=0;
        int numempty=0;
        int i=0;
        while('\0'!=str[i]){
            oldlength++;
            if(' '==str[i]){
                numempty++;
            }
            i++;
        }
        int newlength=oldlength+numempty*2;
        int head=oldlength;
        if(newlength>length)
            return;
        while(head>=0&&newlength>head){
            if(' '==str[head]){
                str[newlength--]='0';
                str[newlength--]='2';
                str[newlength--]='%';
            }
            else{
                str[newlength]=str[head];
                newlength--;

            }
            head--;
        }

    }
};
```

3.从尾到头打印链表

题目描述

输入一个链表，按链表值从尾到头的顺序返回一个ArrayList。

题解

```
/**
*  struct ListNode {
*        int val;
*        struct ListNode *next;
*        ListNode(int x) :
*              val(x), next(NULL) {
*        }
*  };
*/
class Solution {
public:
    vector<int> printListFromTailToHead(ListNode* head) {
        vector<int> res;
        if(head==nullptr)
            return res;
        stack<int> s;
        while(head!=nullptr){
            s.push(head->val);
            head=head->next;
        }
        while(!s.empty()){
            res.push_back(s.top());
            s.pop();
        }
        return res;
    }
};
```

4.重建二叉树

题目描述

输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

题解

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* reConstructBinaryTree(vector<int> pre,vector<int> vin) {
        if(pre.size()==0||vin.size()==0){
            return NULL;
         }
        int aim;
        TreeNode* root=new TreeNode(pre[0]);
        for(int i=0;i<pre.size();i++){
            if(vin[i]==pre[0]){
                aim=i;
                break;
            }

        }
        if(aim==0){
            root->left=NULL;
        }
        else{
            vector<int> tmp_pre,tmp_vin;
            for(int i=1;i<aim+1;i++){
                tmp_pre.push_back(pre[i]);
            }
            for(int i=0;i<aim;i++){
                tmp_vin.push_back(vin[i]);
            }
            root->left=reConstructBinaryTree(tmp_pre,tmp_vin);
        }
        if((aim+1)==vin.size()){
            root->right=NULL;
        }
        else{
            vector<int> tmp_pre,tmp_vin;
            for(int i=aim+1;i<pre.size();i++){
                tmp_pre.push_back(pre[i]);
            }
            for(int i=aim+1;i<vin.size();i++){
                tmp_vin.push_back(vin[i]);
            }
            root->right=reConstructBinaryTree(tmp_pre,tmp_vin);
        }
        return root;
    }
};
```

5.用两个栈实现队列

题目描述

用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

题解

```
class Solution
{
public:
    void push(int node) {
        stack1.push(node);
    }

    int pop() {
        if(stack2.empty()&&stack1.empty())
            return -1;
        if(stack2.empty()&&!stack1.empty()){
            while(!stack1.empty()){
                int tmp=stack1.top();
                stack1.pop();
                stack2.push(tmp);
            }
        }
        int ret=stack2.top();
        stack2.pop();
        return ret;
    }

private:
    stack<int> stack1;
    stack<int> stack2;
};
```

6.旋转数组的最小数字

题目描述

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非减排序的数组的一个旋转，输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。 NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

题解

```
class Solution {
public:
    int minNumberInRotateArray(vector<int> rotateArray) {
        int size=rotateArray.size();
        if(size==0)
            return 0;
        if(size==1)
            return rotateArray[0];
        int low=0,high=size-1;
        while(low<high){
            int mid=low+(high-low)/2;
            if(rotateArray[mid]>rotateArray[high])
                low=mid+1;
            else if(rotateArray[mid]==rotateArray[high])
                high=high-1;
            else{
                high=mid;
            }
        }
        return rotateArray[low];
    }
};
```

7.斐波那契数列

题目描述

大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项（从0开始，第0项为0）。
n<=39

题解

```
class Solution {
public:
    int Fibonacci(int n) {
        if(n==0)
            return 0;
        if(n==1)
            return 1;
        int a=0,b=1;
        while(n>1){
            b=a+b;
            a=b-a;
            n--;
        }
        return b;
    }
};
```

8.跳台阶

题目描述

一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法（先后次序不同算不同的结果）。

题解

```
class Solution {
public:
    int jumpFloor(int number) {
        if(number==0)
            return 0;
        if(number==1)
            return 1;
        if(number==2)
            return 2;
        int a=1,b=2;
        while(number>2){
            b+=a;
            a=b-a;
            number--;
        }
        return b;
    }
};
```

9.变态跳台阶

题目描述

一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

题解

```
class Solution {
public:
    int jumpFloorII(int number) {
        if(number==0)
            return 0;
        if(number>0){
            int total=1;
            total=total<<(number-1);
            return total;
        }
        return -1;
    }
};
```

10.矩形覆盖

题目描述

我们可以用*21的小矩形横着或者竖着去覆盖更大的矩形。请问用n个*21的小矩形无重叠地覆盖一个2*n的大矩形，总共有多少种方法?

题解

```
class Solution {
public:
    int rectCover(int number) {
        if(number<3)
            return number;
        int a=1,b=2;
        while(number>2){
            b+=a;
            a=b-a;
            number--;
        }
        return b;
    }
};
```

11.二进制中1的个数

题目描述

输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

题解

```
class Solution {
public:
     int  NumberOf1(int n) {
         int count=0;

         while(n){
             count++;
            n=(n-1)&n;
         }
         return count;
     }
};
```

12.数值的整数次方

题目描述

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方

题解

```
class Solution {
public:
    double Power(double base, int exponent) {
        if(equal(base,0.0)&&exponent<0)
            return -1;
        unsigned int asbe=(unsigned int)exponent;
        if(exponent<0)
            asbe=(unsigned int )(-exponent);
        double result=absExponent(base,asbe);
        if(exponent<0)
            result=1.0/result;
        return result;
    }
    bool equal(double base,double exponent){
        if(base-exponent<0.0000001&&base-exponent>-0.0000001)
            return true;
        else
            return false;
    }

    double absExponent(double base,unsigned int exponent){
        if(0==exponent)
            return 1;
        if(1==exponent)
            return base;
        double result=absExponent(base,exponent>>1);
        result *=result;
        if(exponent&0x1)
            result*=base;
        return result;
    }

};
```

13.调整数组顺序使奇数位于偶数前面

题目描述

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，使得所有的奇数位于数组的前半部分，所有的偶数位于数组的后半部分，并保证奇数和奇数，偶数和偶数之间的相对位置不变。

题解

```
class Solution {
public:
    void reOrderArray(vector<int> &array) {
        vector<int>::iterator even=array.begin();
        int len= array.size();
        while(len){
            if((*even&0x1)==0){
                int tmp=*even;
                even=array.erase(even);
                array.push_back(tmp);
            }
            else{

                even++;
            }
            len--;
        }
    }
};
```

14.链表中倒数第k个结点

题目描述

输入一个链表，输出该链表中倒数第k个结点。

题解

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* FindKthToTail(ListNode* pListHead, unsigned int k) {
        if(pListHead==NULL||0==k)
            return NULL;
        ListNode* firstnode=pListHead;

        for(unsigned int i=0;i<k-1;i++){
            if(NULL==firstnode->next)
                return NULL;
            firstnode=firstnode->next;

        }
        ListNode* secnode=pListHead;
        while(firstnode->next!=NULL){
            firstnode=firstnode->next;
            secnode=secnode->next;
        }
        return secnode;


    }
};
```

15.反转链表

题目描述

输入一个链表，反转链表后，输出新链表的表头。

题解

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* ReverseList(ListNode* pHead) {
        if(NULL==pHead)
            return NULL;
        ListNode* firNode=pHead;
        ListNode* secNode=pHead->next;
        ListNode* lastNode=pHead->next;
        firNode->next=NULL;
        while(secNode!=NULL){
            lastNode=lastNode->next;
            secNode->next=firNode;
            firNode=secNode;
            secNode=lastNode;

        }
        return firNode;
    }
};
```

16.合并两个排序的链表

题目描述

输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则。

题解

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* Merge(ListNode* pHead1, ListNode* pHead2)
    {
        if(pHead1==NULL)
            return pHead2;
        if(pHead2==NULL)
            return pHead1;
        ListNode* head=NULL;
        ListNode* p1=pHead1;
        ListNode* p2=pHead2;
        if(p1->val<=p2->val){
            head=p1;
            p1=p1->next;
        }
        else{
            head=p2;
            p2=p2->next;
        }
        ListNode* result=head;
        while((p1!=NULL)&&(p2!=NULL)){
            if(p1->val<=p2->val){
                head->next=p1;
                head=head->next;
                p1=p1->next;
            }
            else{
                head->next=p2;
                head=head->next;
                p2=p2->next;
            }

        }
        if(p1!=NULL){
            head->next=p1;
        }
        else if(p2!=NULL){
            head->next=p2;
        }

        return result;
    }
};
```

17.树的子结构

题目描述

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

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    bool HasSubtree(TreeNode* pRoot1, TreeNode* pRoot2)
    {
        bool result=false;
        if((pRoot1!=NULL)&&(pRoot2!=NULL)){
            if(pRoot1->val==pRoot2->val){
                result=DoesTree1HaveTree2(pRoot1,pRoot2);

             }
            if(!result){
                result=HasSubtree(pRoot1->left,pRoot2);
            }
            if(!result){
                result=HasSubtree(pRoot1->right,pRoot2);
            }

         }
        return result;
    }
    bool DoesTree1HaveTree2(TreeNode* pRoot1,TreeNode* pRoot2)
    {
        if(pRoot2==NULL)
            return true;
        if(pRoot1==NULL)
            return false;
        if(pRoot1->val!=pRoot2->val)
            return false;
        return DoesTree1HaveTree2(pRoot1->right,pRoot2->right)
                &&DoesTree1HaveTree2(pRoot1->left,pRoot2->left);
    }
};
```

18.二叉树的镜像

题目描述

操作给定的二叉树，将其变换为源二叉树的镜像。

输入描述:

```
二叉树的镜像定义：源二叉树
            8
           /  \
          6   10
         / \  / \
        5  7 9 11
        镜像二叉树
            8
           /  \
          10   6
         / \  / \
        11 9 7  5
```

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    void Mirror(TreeNode *pRoot) {
        if(pRoot==NULL)
            return;
        swap(pRoot);
        Mirror(pRoot->right);
        Mirror(pRoot->left);
    }
    void swap(TreeNode*& pRoot){
        TreeNode* tmp=pRoot->left;
        pRoot->left=pRoot->right;
        pRoot->right=tmp;
    }

};
```

19.顺时针打印矩阵

题目描述

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，例如，如果输入如下4 X 4矩阵： 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 则依次打印出数字1,2,3,4,8,12,16,15,14,13,9,5,6,7,11,10.

题解

```
class Solution {
public:
    vector<int> printMatrix(vector<vector<int> > matrix) {
        if(matrix.size()==0||matrix[0].size()==0)
            throw "null matrix!";
        int start=0;
        vector<int> res;
        while(((int)(matrix.size()))>(start*2)&&((int)(matrix[0].size()))>(start*2)){
            printCircle(matrix,start,res);
            start++;
        }
        return res;
    }
    void printCircle(vector<vector<int>> matrix,int start,vector<int>& res)
    {
        int endy=matrix.size()-1-start;
        int endx=matrix[0].size()-1-start;
        for(int i=start;i<=endx;i++){
            int number=matrix[start][i];
            res.push_back(number);
        }
        for(int i=start+1;i<=endy;i++){
            int number=matrix[i][endx];
            res.push_back(number);
        }
        if(start<endy)//防止打印第一行（只有第一行的情况下）
        for(int i=endx-1;i>=start;i--){
            int number=matrix[endy][i];
            res.push_back(number);
        }

        if(start<endx)//防止打印第一列（只有第一列的情况下）
        for(int i=endy-1;i>start;i--){
            int number=matrix[i][start];
            res.push_back(number);
        }
    }
};
```

20.包含min函数的栈

题目描述

定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的min函数（时间复杂度应为O（1））。

题解

```
class Solution {
public:
    stack<int> stack1,stack2;
    void push(int value) {
        stack1.push(value);
        if(stack2.empty())
            stack2.push(value);
        else if(stack2.top()>value)
            stack2.push(value);
        else
            stack2.push(stack2.top());

    }
    void pop() {
        stack1.pop();
        stack2.pop();
    }
    int top() {
        return stack1.top();
    }
    int min() {
        return stack2.top();
    }
};
```

21.栈的压入、弹出序列

题目描述

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否可能为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，序列4,5,3,2,1是该压栈序列对应的一个弹出序列，但4,3,5,1,2就不可能是该压栈序列的弹出序列。（注意：这两个序列的长度是相等的）

题解

```
class Solution {
public:
    bool IsPopOrder(vector<int> pushV,vector<int> popV) {
        if(pushV.empty()||popV.empty())
            return false;
        stack<int> stackV;
        bool flag=false;
        int pus=0;
        stackV.push(pushV[pus++]);
        int i=0;
        while(pus<=popV.size()){
            if(stackV.top()==popV[i]){
                stackV.pop();
                i++;
                if(i==pushV.size()){
                    flag=true;
                    break;
                }
            }
            else{
                if(pus==pushV.size()){
                    flag=false;
                    break;
                }
                while(stackV.top()!=popV[i]&&pus<=popV.size()){
                    stackV.push(pushV[pus++]);
                }

            }
        }
        return flag;
    }
};
```

22.从上往下打印二叉树

题目描述

从上往下打印出二叉树的每个节点，同层节点从左至右打印。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    vector<int> PrintFromTopToBottom(TreeNode* root) {
        vector<int> res;
        if(root==NULL)
            return res;
        queue<TreeNode*> queueTreeNode;

        queueTreeNode.push(root);
        while(!queueTreeNode.empty()){
            root=queueTreeNode.front();
            queueTreeNode.pop();
            res.push_back(root->val);
            if(root->left!=NULL){
                queueTreeNode.push(root->left);
            }
            if(root->right!=NULL){
                queueTreeNode.push(root->right);
            }

        }
        return res;
    }
};
```

23.二叉搜索树的后序遍历序列

题目描述

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出Yes,否则输出No。假设输入的数组的任意两个数字都互不相同。

题解

```
class Solution {
public:
    bool VerifySquenceOfBST(vector<int> sequence) {
        if(sequence.empty())
            return false;
        int l=0;
        int root=sequence[sequence.size()-1];
        for(;l<sequence.size()-1;l++){
            if(sequence[l]>root)
                break;
        }
        int r=l;
        for(;r<sequence.size()-1;r++){
            if(sequence[r]<root)
                return false;
        }
        bool left=true;
        if(l>0){
            vector<int> leftBST;
            for(int i=0;i<l;i++)
                leftBST.push_back(sequence[i]);
            left=VerifySquenceOfBST(leftBST);
        }
        bool right=true;
        if(r<sequence.size()-1){
            vector<int> rightBST;
            for(int i=l;i<sequence.size()-1;i++)
                rightBST.push_back(sequence[i]);
            right=VerifySquenceOfBST(rightBST);
        }
        return (left&&right);
    }
};
```

24.二叉树中和为某一值的路径

题目描述

输入一颗二叉树的跟节点和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。路径定义为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径。(注意: 在返回值的list中，数组长度大的数组靠前)

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    vector<vector<int> > FindPath(TreeNode* root,int expectNumber) {
        vector<int> res;
        int total=0;
        vector<vector<int>> result;
        Findpath_all(root,expectNumber,res,total,result);
        return result;
    }
    void Findpath_all(TreeNode* root,int expectNumber,vector<int>& res,
                                    int & total,vector<vector<int>>& result)
    {
        if(root!=NULL){
        res.push_back(root->val);
        total+=root->val;
        if(expectNumber==total&&root->right==NULL&&root->left==NULL){
            result.push_back(res);
        }
        if(NULL!=root->left){
            Findpath_all(root->left,expectNumber,res,total,result);
        }
        if(NULL!=root->right){
            Findpath_all(root->right,expectNumber,res,total,result);
        }
        res.pop_back();
        total-=root->val;
        }
    }
};
```

25.复杂链表的复制

题目描述

输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），返回结果为复制后复杂链表的head。（注意，输出结果中请不要返回参数中的节点引用，否则判题程序会直接返回空）

题解

```
/*
struct RandomListNode {
    int label;
    struct RandomListNode *next, *random;
    RandomListNode(int x) :
            label(x), next(NULL), random(NULL) {
    }
};
*/
class Solution {
public:
    RandomListNode* Clone(RandomListNode* pHead)
    {
        ClonenextNodes(pHead);
        ClonerandomNodes(pHead);
        return ReconnectNodes(pHead);
    }
    void ClonenextNodes(RandomListNode* pHead)
    {
        if(pHead==NULL)
            return;
        RandomListNode* node=pHead;
        while(node!=NULL){
            RandomListNode* newNode=new RandomListNode(node->label);
            newNode->next=node->next;
            newNode->label=node->label;
            newNode->random=NULL;
            node->next=newNode;
            node=newNode->next;
        }
    }
    void ClonerandomNodes(RandomListNode* pHead)
    {
        if(pHead==NULL)
            return;

        RandomListNode* head=pHead;
        while(head!=NULL){
            RandomListNode* node=head->next;
            if(head->random!=NULL)
                 node->random=head->random->next;
            head=node->next;
        }
    }
    RandomListNode* ReconnectNodes(RandomListNode* pHead)
    {
        if(pHead==NULL)
            return NULL;
        RandomListNode* head=pHead->next;

        RandomListNode* tmp=pHead;
        RandomListNode* clone=tmp->next;

        tmp->next=clone->next;
```

```
        tmp=tmp->next;
        while(tmp!=NULL){
            clone->next=tmp->next;
            clone=clone->next;
            tmp->next=clone->next;
            tmp=tmp->next;

        }

        return head;
    }
};
```

26.二叉搜索树与双向链表

题目描述

输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的双向链表。要求不能创建任何新的结点，只能调整树中结点指针的指向。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    TreeNode* Convert(TreeNode* pRootOfTree)
    {
        if(pRootOfTree==NULL)
            return NULL;
        TreeNode* LastNode=NULL;
        ConvertNode(pRootOfTree,LastNode);
        if(LastNode!=NULL)
            while(LastNode->left!=NULL)
                LastNode=LastNode->left;
        return LastNode;
    }
    void ConvertNode(TreeNode* pRootOfTree,TreeNode*& LastNode)
    {
        if(pRootOfTree==NULL)
            return;
        if(pRootOfTree->left!=NULL)
            ConvertNode(pRootOfTree->left,LastNode);
        pRootOfTree->left=LastNode;
        if(LastNode!=NULL){
            LastNode->right=pRootOfTree;
        }
        LastNode=pRootOfTree;

        if(pRootOfTree->right!=NULL)
            ConvertNode(pRootOfTree->right,LastNode);

    }
};
```

27.字符串的排列

题目描述

输入一个字符串,按字典序打印出该字符串中字符的所有排列。例如输入字符串abc,则打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。

输入描述:

> 输入一个字符串,长度不超过9(可能有字符重复),字符只包括大小写字母。

题解

```
class Solution {
public:
    vector<string> Permutation(string str) {
        vector<string> res;
        if(!str.empty())
            Permutation_All(str,0,res);
        sort(res.begin(),res.end());
        return res;
    }
    void Permutation_All(string& str,int k,vector<string>& res)
    {
        if(k==str.length())
            res.push_back(str);
        else{
            for(int i=k;i<str.length();i++){
                if(i==k||str[i]!=str[k]){
                char tmp=str[i];
                str[i]=str[k];
                str[k]=tmp;

                Permutation_All(str,k+1,res);
                tmp=str[i];
                str[i]=str[k];
                str[k]=tmp;
                }
            }

        }
    }
};
```

28.数组中出现次数超过一半的数字

题目描述

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。如果不存在则输出0。

题解

```
class Solution {
public:
    int MoreThanHalfNum_Solution(vector<int> numbers) {
        if(numbers.empty())
            return 0;
        int result;
        int times=0;
        for(int i=0;i<numbers.size();i++){
            if(times==0){
                result=numbers[i];
                times=1;
            }
            else if(numbers[i]==result){
                times++;
            }
            else{
                times--;
            }
        }
        if(CheckResult(numbers,result))
            result=0;
        return result;

    }
    bool CheckResult(vector<int> numbers,int result)
    {
        int times=0;
        bool check=false;
        for(int i=0;i<numbers.size();i++)
        {
            if(numbers[i]==result)
                times++;
        }
        if(times*2<=numbers.size())
            check=true;
        return check;
    }
};
```

29.最小的K个数

题目描述

输入n个整数，找出其中最小的K个数。例如输入4,5,1,6,2,7,3,8这8个数字，则最小的4个数字是1,2,3,4,。

题解

```
class Solution {
public:
    typedef multiset<int,greater<int>> inSet;
    typedef multiset<int,greater<int>>::iterator setIter;
    vector<int> GetLeastNumbers_Solution(vector<int> input, int k) {
        vector<int> res;
        if(input.empty()||input.size()<k)
            return res;
        inSet setNum;
        setNum.clear();

        for(int i=0;i<input.size();i++){
            if(setNum.size()<k){
                setNum.insert(input[i]);
            }
            else{
                setIter Ibegin=setNum.begin();
                if(input[i]<*Ibegin){
                    setNum.erase(Ibegin);
                    setNum.insert(input[i]);
                }
            }
        }

        for(setIter Ibegin=setNum.begin();Ibegin!=setNum.end();++Ibegin){
            res.push_back(*Ibegin);
        }
        sort(res.begin(),res.end());
        return res;
    }
};
```

30.连续子数组的最大和

题目描述

HZ偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后,他又发话了:在古老的一维模式识别中,常常需要计算连续子向量的最大和,当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,是否应该包含某个负数,并期望旁边的正数会弥补它呢？例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为8(从第0个开始,到第3个为止)。给一个数组，返回它的最大连续子序列的和，你会不会被他忽悠住？(子向量的长度至少是1)

题解

```
class Solution {
public:
    int FindGreatestSumOfSubArray(vector<int> array) {
        if(array.empty())
           return 0;
        int sum=array[0];
        int max=array[0];
        for(int i=1;i<array.size();i++){
            if(sum<=0){
                sum=array[i];
                if(sum>max)
                    max=sum;
            }
            else{
                sum+=array[i];
                if(sum>max){
                    max=sum;
                }
            }
        }
        return max;
    }
};
```

31.整数中1出现的次数（从1到n整数中1出现的次数）

题目描述

求出1~13的整数中1出现的次数,并算出100~1300的整数中1出现的次数？为此他特别数了一下1~13中包含1的数字有1、10、11、12、13因此共出现6次,但是对于后面问题他就没辙了。ACMer希望你们帮帮他,并把问题更加普遍化,可以很快的求出任意非负整数区间中1出现的次数（从1 到 n 中1出现的次数）。

题解

```
class Solution {
public:
    int NumberOf1Between1AndN_Solution(int n)
    {
        int res=0;
        for(int i=1;i<=n;i*=10){
            int a=n/i,b=n%i;
            if(a%10==1){
                res+=a/10*i+b+1;
            }
            else{
                res+=(a+8)/10*i;
            }
        }
        return res;
    }
};
```

32.把数组排成最小的数

题目描述

输入一个正整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。例如输入数组{3，32，321}，则打印出这三个数字能排成的最小数字为321323。

题解

```
class Solution {
public:
    string PrintMinNumber(vector<int> numbers) {
        string res;
        if(numbers.empty())
            return res;
        vector<string> strNumbers;
        for(int i=0;i<numbers.size();i++){
            strNumbers.push_back(to_string(numbers[i]));
        }
        sort(strNumbers.begin(),strNumbers.end(),compare);
        res=strNumbers[0];
        for(int i=1;i<strNumbers.size();i++){
            res+=strNumbers[i];
        }
        return res;
    }
    static int compare(string& a,string& b)
    {
        string ab=a+b;
        string ba=b+a;
        return (ab<ba);
    }
};
```

33.丑数

题目描述

把只包含质因子2、3和5的数称作丑数（Ugly Number）。例如6、8都是丑数，但14不是，因为它包含质因子7。习惯上我们把1当做是第一个丑数。求按从小到大的顺序的第N个丑数。

题解

```
class Solution {
public:
    int GetUglyNumber_Solution(int index) {
        if(index<=0)
               return 0;
        int *resNum=new int[index];
        resNum[0]=1;
        int next=1;
        int *ugly2=resNum;
        int *ugly3=resNum;
        int *ugly5=resNum;
        while(next<index){
            int min=Min(*ugly2*2,*ugly3*3,*ugly5*5);
            resNum[next]=min;
            while(*ugly2*2<=min)
                ++ugly2;
            while(*ugly3*3<=min)
                ++ugly3;
            while(*ugly5*5<=min)
                ++ugly5;
            next++;
        }
        int res=resNum[next-1];
        delete[] resNum;
        return res;
    }
    int Min(int u2,int u3,int u5)
    {
        int min=(u2<u3)?u2:u3;
        min=(min<u5)?min:u5;
        return min;
    }
};
```

34.第一个只出现一次的字符

题目描述

在一个字符串(0<=字符串长度<=10000，全部由字母组成)中找到第一个只出现一次的字符,并返回它的位置, 如果没有则返回 -1（需要区分大小写）.

题解

```
class Solution {
public:
    int FirstNotRepeatingChar(string str) {
        if(str.empty())
            return -1;
        int strNum[256]={0};
        int len=str.length();
        int res=-1;
        for(int i=0;i<len;i++){
            int tmp=str[i];
            strNum[tmp]++;
        }

        for(int i=0;i<len;i++){
            if(strNum[(int)(str[i])]==1){
                res=i;
                break;
            }
        }
        return res;

    }
};
```

35.数组中的逆序对

题目描述

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组,求出这个数组中的逆序对的总数P。并将P对1000000007取模的结果输出。 即输出P%1000000007

输入描述:

题目保证输入的数组中没有的相同的数字数据范围：　对于%50的数据,size<=10^4　对于%75的数据,size<=10^5　对于%100的数据,size<=2*10^5

示例1

输入

复制

```
1,2,3,4,5,6,7,0
```

输出

复制

```
7
```

题解

```
class Solution {
public:
    int InversePairs(vector<int> data) {
        if(data.empty())
            return -1;
        vector<int> copy;
        int len=data.size();
        for(int i=0;i<len;i++)
            copy.push_back(data[i]);
        int count= InversePairsCore(data,0,len-1,copy);
        count=count%1000000007;

        return count;
    }

    int InversePairsCore(vector<int>& data,int start,int end,vector<int>& copy)
    {
        if(start==end){
            copy[start]=data[start];
            return 0;
        }
        int mid=(end-start)/2;
        /*对已经统计了逆序对的数组，我们需要对其排好序，以避免在以后的统计过程中再次重复统计，
        所以copy数组就是起到这个作用，当然，这里的有序只是“局部有序”，整体来看还是无序的。
        既然copy数组是“有序”的，下一次就直接在这个基础上进行统计就可以，原始数据data用来充
        当原来copy数组的角色来保存“更加有序”的数组，这样进行递归，就节省了数据来回拷贝所浪费的时间。
        */
        int left=InversePairsCore(copy,start,start+mid,data)%1000000007;
        int right=InversePairsCore(copy,start+mid+1,end,data)%1000000007;
        int m=start+mid;
        int j=end;
        int index=end;
        int count=0;
        while(m>=start&&j>=start+mid+1){
            if(data[m]>data[j]){
                copy[index--]=data[m--];
                count+=j-(start+mid);
                if(count>=1000000007)
                    count%=1000000007;
            }
            else{
                copy[index--]=data[j--];
            }
        }
        while(m>=start){
            copy[index--]=data[m--];
        }
        while(j>=start+mid+1)
            copy[index--]=data[j--];

        return (count+right+left)%1000000007;
    }
```

```
};
```

36.两个链表的第一个公共结点

题目描述

输入两个链表，找出它们的第一个公共结点。

题解

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* FindFirstCommonNode( ListNode* pHead1, ListNode* pHead2) {
        if(pHead1==NULL||pHead2==NULL)
            return NULL;

        int len1=0,len2=0;
        ListNode* p1=pHead1;
        ListNode* p2=pHead2;
        while(p1!=NULL){
            len1++;
            p1=p1->next;

        }
        while(p2!=NULL){
            len2++;
            p2=p2->next;
        }
        p1=pHead1;
        p2=pHead2;
        if(len1>=len2){
            int tmp=len1-len2;
            for(;tmp>0;tmp--){
                p1=p1->next;
            }
        }
        else{
            int tmp=len2-len1;
            for(;tmp>0;tmp--){
                p2=p2->next;
            }
        }
        while(p1!=NULL&&p2!=NULL){
            if(p1==p2){
                return p1;
            }
            p1=p1->next;
            p2=p2->next;
        }
        return NULL;

    }
};
```

37.数字在排序数组中出现的次数

题目描述

统计一个数字在排序数组中出现的次数。

题解

```
class Solution {
public:
    int GetNumberOfK(vector<int> data ,int k) {
        if(data.empty())
            return 0;
        int len=data.size();
        int first=GetFirstk(data,k,0,len-1);
        int last=GetLastk(data,k,0,len-1);
        if(last<0||first<0)
            return 0;
        return (last-first+1);
    }
    int GetFirstk(vector<int>& data,int k,int start,int end)
    {
        if(start>end)
            return -1;
        /*mid=(start+end)/2，这是一个存在潜在风险的写法，因为如果数组长度非常大，
        start+end就有可能会溢出，所以写法改为 mid=start+(end-start)/2，才算完美。
        */
        int mid=start+(end-start)/2;
        int tmp=0;
        while(start<=end){
           tmp=data[mid];
           if(tmp==k){
               if((mid>0&&data[mid-1]!=k)||mid==0){
                    return mid;
                }
                else
                    end=mid-1;
            }
            else if(tmp>k){
                end=mid-1;
            }
            else{
                start=mid+1;
            }
            mid=start+(end-start)/2;
        }
        return -1;
    }
    int GetLastk(vector<int>& data,int k,int start,int end)
    {
        if(start>end)
            return -1;
        int mid=start+(end-start)/2;
        int tmp=0;
        int len=data.size()-1;
        while(start<=end){
            tmp=data[mid];
            if(tmp==k){
                if((mid<len&&data[mid+1]!=k)||mid==len){
                    return mid;

                }
```

```
                else
                    start=mid+1;
            }
            else if(tmp>k){
                end=mid-1;
            }
            else{
                start=mid+1;
            }
            mid=start+(end-start)/2;
        }
        return -1;
    }
};
```

38.二叉树的深度

题目描述

输入一棵二叉树，求该树的深度。从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    int TreeDepth(TreeNode* pRoot)
    {
        if(pRoot==NULL){
            return 0;
        }
        int left=TreeDepth(pRoot->left);
        int right=TreeDepth(pRoot->right);
        return (left>right)?(left+1):(right+1);
    }
};
```

39.平衡二叉树

题目描述

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

题解

```
class Solution {
public:
    bool IsBalanced_Solution(TreeNode* pRoot) {
        int dep=0;
        return IsBalanced(pRoot,&dep);
    }
    bool IsBalanced(TreeNode* pRoot,int *dep)
    {
        if(pRoot==NULL){
            *dep=0;
            return true;
        }
        int left=0,right=0;
        if(IsBalanced(pRoot->left,&left)&&IsBalanced(pRoot->right,&right)){
            int diff=left-right;
            if(diff<=1&&diff>=-1){
                *dep=1+((right>left)?right:left);
                return true;
            }
        }
        return false;
    }
};
```

40.数组中只出现一次的数字

题目描述

一个整型数组里除了两个数字之外，其他的数字都出现了偶数次。请写程序找出这两个只出现一次的数字。

题解

```
class Solution {
public:
    void FindNumsAppearOnce(vector<int> data,int* num1,int *num2) {
        *num1=*num2=0;
        if(data.empty())
                return;
        int res=0;
        int len=data.size();
        for(int i=0;i<len;i++){
            res^=data[i];
        }
        int index=findIndex(res);
        if(index >= 8*sizeof(int))
            return;
        for(int i=0;i<len;i++){
            if(checkbit(data[i],index))
                *num1^=data[i];
            else
                *num2^=data[i];
        }
        if(*num1>*num2){
            int tmp=*num1;
            *num1=*num2;
            *num2=tmp;
        }

    }
    int findIndex(int res){
        int ret=0;
        while(((res & 1)==0)&&(ret<8*sizeof(int))){
            res=res>>1;
            ret++;
        }
        return ret;
    }
    bool checkbit(int data,int index)
    {
        data=data>>index;
        return (data & 1);
    }
};
```

41.和为S的连续正数序列

题目描述

小明很喜欢数学,有一天他在做数学作业时,要求计算出9~16的和,他马上就写出了正确答案是100。但是他并不满足于此,他在想究竟有多少种连续的正数序列的和为100(至少包括两个数)。没多久,他就得到另一组连续正数和为100的序列:18,19,20,21,22。现在把问题交给你,你能不能也很快的找出所有和为S的连续正数序列? Good Luck!

输出描述:

输出所有和为S的连续正数序列。序列内按照从小至大的顺序，序列间按照开始数字从小到大的顺序

题解

```
class Solution {
public:
    vector<vector<int> > FindContinuousSequence(int sum) {
        vector<vector<int> > res;
        if(sum<3)
            return res;
        int small=1;
        int big=2;
        int middle=(1+sum)/2;
        int total=small+big;

        while(small<middle){
            vector<int> tmp;
            if(total==sum){
                for(int i=small;i<=big;i++){
                    tmp.push_back(i);
                }
                big++;
                total+=big;
                res.push_back(tmp);
            }
            else{
                if(total<sum){
                    big++;
                    total+=big;
                }
                else{
                    total-=small;
                    small++;
                }
            }
        }
        return res;
    }
};
```

42.和为S的两个数字

题目描述

输入一个递增排序的数组和一个数字S，在数组中查找两个数，使得他们的和正好是S，如果有多对数字的和等于S，输出两个数的乘积最小的。

输出描述:

对应每个测试案例，输出两个数，小的先输出。

题解

```
class Solution {
public:
    vector<int> FindNumbersWithSum(vector<int> array,int sum) {
        vector<int> res;
        if(array.empty())
            return res;
        int begin=0;
        int len=array.size();
        int end=len-1;
        int total=array[begin]+array[end];
        while(begin<end){
            if(sum==total){
                res.push_back(array[begin]);
                res.push_back(array[end]);
                break;
            }
            else{
                if(total>sum){
                    end--;
                    total=array[begin]+array[end];
                }
                else{
                    begin++;
                    total=array[begin]+array[end];
                }
            }
        }
        return res;
    }
};
```

43.左旋转字符串

题目描述

汇编语言中有一种移位指令叫做循环左移（ROL），现在有个简单的任务，就是用字符串模拟这个指令的运算结果。对于一个给定的字符序列S，请你把其循环左移K位后的序列输出。例如，字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。是不是很简单？OK，搞定它！

题解

```
class Solution {
public:
    string LeftRotateString(string str, int n) {
        if(str.empty())
            return str;
        Reserve(str,0,n-1);
        Reserve(str,n,str.size()-1);
        Reserve(str,0,str.size()-1);
        return str;
    }
    void Reserve(string & str,int begin,int end)
    {
        if(str.empty())
            return;
        int tmp=0;
        while(begin<end){
            tmp=str[begin];
            str[begin]=str[end];
            str[end]=tmp;
            begin++;
            end--;
        }
    }
};
```

44.翻转单词顺序列

题目描述

最近来了一个新员工Fish，每天早晨总是会拿着一本英文杂志，写些句子在本子上。同事Cat对Fish写的内容颇感兴趣，有一天他向Fish借来翻看，但却读不懂它的意思。例如，“student. a am I”。后来才意识到，这家伙原来把句子单词的顺序翻转了，正确的句子应该是“I am a student.”。Cat对一一的翻转这些单词顺序可不在行，你能帮助他么?

题解

```
class Solution {
public:
    string ReverseSentence(string str) {
        if(str.empty())
            return str;
        int begin=0;
        int end=str.size()-1;
        Reserve(str,begin,end);
        end=0;
        while(begin<str.size()){
            if(str[begin]==' '){
                begin++;
                end++;
            }
            else if(end==(str.size())||(str[end]==' ')){
                Reserve(str,begin,--end);
                begin=++end;
            }
            else{
                end++;
            }

        }
        return str;
    }
    void Reserve(string & str,int begin,int end)
    {
        if(str.empty())
            return;
        int tmp=0;
        while(begin<end){
            tmp=str[begin];
            str[begin]=str[end];
            str[end]=tmp;
            begin++;
            end--;
        }
    }
};
```

45.扑克牌顺子

题目描述

LL今天心情特别好,因为他去买了一副扑克牌,发现里面居然有2个大王,2个小王(一副牌原本是54张^_^)...他随机从中抽出了5张牌,想测测自己的手气,看看能不能抽到顺子,如果抽到的话,他决定去买体育彩票,嘿嘿！！“红心A,黑桃3,小王,大王,方片5”,“Oh My God!”不是顺子.....LL不高兴了,他想了想,决定大\小 王可以看成任何数字,并且A看作1,J为11,Q为12,K为13。上面的5张牌就可以变成“1,2,3,4,5”(大小王分别看作2和4),“So Lucky!”。LL决定去买体育彩票啦。 现在,要求你使用这幅牌模拟上面的过程,然后告诉我们LL的运气如何， 如果牌能组成顺子就输出true，否则就输出false。为了方便起见,你可以认为大小王是0。

题解

```
class Solution {
public:
    bool IsContinuous( vector<int> numbers ) {
        if(numbers.empty())
            return false;
        int numzeros=0;
        int gap=0;
        sort(numbers.begin(),numbers.end());
        for(int i=0;i<numbers.size()&&numbers[i]==0;i++){
            numzeros++;
        }
        int small=numzeros;
        int big=small+1;
        while(big<numbers.size()){
            if(numbers[small]==numbers[big]){
                return false;//重复
            }
            gap+=numbers[big]-numbers[small]-1;
            small++;
            big++;
        }
        return ((gap>numzeros)?false:true);
    }
};
```

46.孩子们的游戏(圆圈中最后剩下的数)

题目描述

有个游戏是这样的:首先,让小朋友们围成一个大圈。然后,他随机指定一个数m,让编号为0的小朋友开始报数。每次喊到m-1的那个小朋友要出列唱首歌,然后可以在礼品箱中任意的挑选礼物,并且不再回到圈中,从他的下一个小朋友开始,继续0...m-1报数....这样下去....直到剩下最后一个小朋友,可以不用表演,并且拿到“名侦探柯南”典藏版(名额有限哦!!^_^)。请你试着想下,哪个小朋友会得到这份礼品呢？(注：小朋友的编号是从0到n-1)

题解

```
class Solution {
public:
    int LastRemaining_Solution(int n, int m)
    {
        if(n<1||m<1)
            return -1;
        int last=0;
        for(int i=2;i<=n;i++){
            last=(last+m)%i;
        }
        return last;
    }
};
```

47.求1+2+3+...+n

题目描述

求1+2+3+...+n，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

题解

```
class Solution {
public:
    int Sum_Solution(int n) {
        n&&(n+=Sum_Solution(n-1));
        return n;
    }
};
```

48.不用加减乘除做加法

题目描述

写一个函数，求两个整数之和，要求在函数体内不得使用+、-、*、/四则运算符号

题解

```
class Solution {
public:
    int Add(int num1, int num2)
    {
        int sum,a;
        do{
            sum=num1^num2;
            a=(num1&num2)<<1;
            num1=sum;
            num2=a;
        }while(num2!=0);
        return num1;
    }
};
```

49.把字符串转换成整数

题目描述

将一个字符串转换成一个整数(实现Integer.valueOf(string)的功能，但是string不符合数字要求时返回0)，要求不能使用字符串转换整数的库函数。 数值为0或者字符串不是一个合法的数值则返回0。

输入描述:

```
输入一个字符串,包括数字字母符号,可以为空
```

输出描述:

```
如果是合法的数值表达则返回该数字，否则返回0
```

示例1

输入

```
+2147483647
    1a33
```

输出

```
2147483647
    0
```

题解

```
class Solution {
public:
    int StrToInt(string str) {
        if(str.empty())
            return 0;
        int res=0;
        int flag=0;//正数
        int i=0;
        if(str[0]=='-'){
            flag=1;
            i++;
        }
        else if(str[0]=='+'){
            flag=0;
            i++;
        }
        else
            flag=2;//前面没有符号，正数
        for(;i<str.size();i++){
            res*=10;
            int tmp=str[i]-'0';
            if(tmp<0||tmp>9)
                return 0;
            res+=tmp;
            if((flag==1&&res<INT_MIN)||res>INT_MAX)
                return 0;
        }
        if(flag==1)
            res=-res;
        return res;
    }
};
```

50.数组中重复的数字

题目描述

在一个长度为n的数组里的所有数字都在0到n-1的范围内。 数组中某些数字是重复的，但不知道有几个数字是重复的。也不知道每个数字重复几次。请找出数组中任意一个重复的数字。 例如，如果输入长度为7的数组{2,3,1,0,2,5,3}，那么对应的输出是第一个重复的数字2。

题解

```
class Solution {
public:
    // Parameters:
    //        numbers:     an array of integers
    //        length:      the length of array numbers
    //        duplication: (Output) the duplicated number in the array number
    // Return value:       true if the input is valid, and there are some duplications
in the array number
    //                     otherwise false
    bool duplicate(int numbers[], int length, int* duplication) {
        if(numbers==nullptr||length<=0)
            return false;
        for(int i=0;i<length;i++){
            if(numbers[i]<0||numbers[i]>length-1)
                return false;
        }
        for(int i=0;i<length;i++){
            while(numbers[i]!=i){
                if(numbers[i]==numbers[numbers[i]]){
                    *duplication=numbers[i];
                    return true;
                }
                int tmp=numbers[i];
                numbers[i]=numbers[tmp];
                numbers[tmp]=tmp;
            }
        }
        return false;
    }
};
```

51.构建乘积数组

题目描述

给定一个数组A[0,1,...,n-1],请构建一个数组B[0,1,...,n-1],其中B中的元素B[i]=A[0]*A[1]*...*A[i-1]*A[i+1]...A[n-1]。不能使用除法。

题解

```
class Solution {
public:
    /*求B[i]=A[0]*A[1]*...*A[n-1]
    动态规划:
    B[0]=1;
    for(int i=1;i<len;i++){
        B[i]=B[i-1]*A[i-1];
    }
    */
    vector<int> multiply(const vector<int>& A) {
        vector<int> B;

        if(A.size()>0){
            for(int i=0;i<A.size();i++){
                B.push_back(1);
            }
            int tmp=1;
            for(int i=1;i<A.size();i++){//C[i]=c[i-1]*A[i-1]  C==B
                B[i]=B[i-1]*A[i-1];
            }
            for(int i=A.size()-2;i>=0;i--){
                tmp*=A[i+1];//D[i]=D[i+1]*A[i+1]
                B[i]=B[i]*tmp;//C[i]*D[i]
            }
        }
        return B;
    }
};
```

52.正则表达式匹配

题目描述

请实现一个函数用来匹配包括'.'和'*'*的正则表达式。模式中的字符'.'表示任意一个字符，而*表示它前面的字符可以出现任意次（包含0次）。 在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab*ac*a"匹配，但是与"aa.a"和"ab*a"均不匹配

题解

```
class Solution {
public:
    bool match(char* str, char* pattern)
    {
        if(str==nullptr||pattern==nullptr)
            return false;
        return matchCore(str,pattern);
    }

    bool matchCore(char* str, char* pattern)
    {
        if(*str=='\0'&&*pattern=='\0')
            return true;
        if(*str!='\0'&&*pattern=='\0')
            return false;
        if(*(pattern+1)=='*'){
            if(*str==*pattern||(*pattern=='.'&&*str!='\0')){//当前指向的char匹配
                return matchCore(str+1,pattern+2)|| //*只匹配一次
                    matchCore(str+1,pattern)|| //*匹配多次
                    matchCore(str,pattern+2); //*匹配0次

            }
            else
                return matchCore(str,pattern+2);//*匹配0次
        }

        if(*str==*pattern||(*pattern=='.'&&*str!='\0'))//当前指向的char匹配
            return matchCore(str+1,pattern+1);

        return false;

    }
};
```

53.表示数值的字符串

题目描述

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100","5e2","-123","3.1416"和"-1E-16"都表示数值。 但是"12e","1a3.14","1.2.3","+-5"和"12e+4.3"都不是。

题解

```
class Solution {
public:
    bool isNumeric(char* string)
    {
        if(string==nullptr)
            return false;
        bool flag=scansymbol(string);
        if(*string=='.'){
            string++;
            flag=scannumber(string)||flag;//只有小数点前后没有数字时才为false
        }
        if(*string=='e'||*string=='E'){
            string++;
            flag=flag&&scansymbol(string);//只有e|E前后有数字时才为true
        }
        return (flag&&*string=='\0');
    }
    bool scansymbol(char*& string)
    {
        if(*string=='+'||*string=='-')
            ++string;
         return scannumber(string);
    }
    bool scannumber(char*& string)
    {
        char * head=string;
        while(string!=nullptr&&*string>='0'&&*string<='9')
            string++;
        return string>head;
    }
};
```

54.字符流中第一个不重复的字符

题目描述

请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符"go"时，第一个只出现一次的字符是"g"。当从该字符流中读出前六个字符“google"时，第一个只出现一次的字符是"l"。

输出描述:

如果当前字符流没有存在出现一次的字符，返回#字符。

题解

```
class Solution
{
private:

public:
  //Insert one char from stringstream
    char res[256]={0};
    string s;
    void Insert(char ch)
    {
        s+=ch;
        res[ch]++;

    }
  //return the first appearence once char in current stringstream
    char FirstAppearingOnce()
    {
        for(int i=0;i<s.size();i++){
            if(res[s[i]]==1)
                return s[i];
        }
        return '#';
    }

};
```

55.链表中环的入口结点

题目描述

给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出null。

题解

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
        val(x), next(NULL) {
    }
};
*/
class Solution {
public:
    ListNode* EntryNodeOfLoop(ListNode* pHead)
    {
        if(pHead==nullptr)
            return nullptr;
        ListNode* p1=pHead->next;
        if(p1==nullptr)
            return nullptr;
        ListNode* p2=p1->next;
        while(p1!=nullptr&&p2!=nullptr&&p2!=p1){//判断是否存在环
            p1=p1->next;
            p2=p2->next;
            if(p2!=nullptr)
                p2=p2->next;
        }
        if(p1==nullptr||p2==nullptr)
            return nullptr;
        int num=1;
        p2=p2->next;
        while(p2!=p1){//得出环中结点数目
            num++;
            p2=p2->next;
        }
        p1=pHead;
        p2=pHead;
        for(int i=0;i<num;i++){//p2先从头走n步
            p2=p2->next;
        }
        while(p2!=p1){//找到入口结点
            p2=p2->next;
            p1=p1->next;
        }
        return p1;
    }
};
```

56.删除链表中重复的结点

题目描述

在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 例如，链表1->2->3->3->4->4->5 处理后为 1->2->5

题解

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
        val(x), next(NULL) {
    }
};
*/
class Solution {
public:
    ListNode* deleteDuplication(ListNode* pHead)
    {
        if(pHead==NULL)
            return NULL;
        ListNode* f=pHead;
        ListNode* l=pHead;
        ListNode* m=pHead;
        while(l!=NULL){
            if(l->next!=NULL&&l->next->val==f->val){
                l=l->next;
            }
            else if(f!=l){
                if(f==pHead){
                    l=l->next;
                    f=l;
                    m=l;
                    pHead=l;

                }
                else{
                    m->next=l->next;
                    l=l->next;
                    f=l;
                }
            }
            else{
                m=f;
                f=f->next;
                l=l->next;

            }
        }
        return pHead;
    }
};
```

57.二叉树的下一个结点

题目描述

给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

题解

```
/*
struct TreeLinkNode {
    int val;
    struct TreeLinkNode *left;
    struct TreeLinkNode *right;
    struct TreeLinkNode *next;
    TreeLinkNode(int x) :val(x), left(NULL), right(NULL), next(NULL) {

    }
};
*/
class Solution {
public:
    TreeLinkNode* GetNext(TreeLinkNode* pNode)
    {
        if(pNode==nullptr)
            return nullptr;
        TreeLinkNode* res=nullptr;
       if(pNode->right!=nullptr){//如果右子树存在，那么下一个结点在右子树中
           pNode=pNode->right;
           while(pNode->left!=nullptr)
               pNode=pNode->left;
           res=pNode;

       }
       else{//如果右子树不存在，那么下一个结点在父结点中或者不存在
           while(pNode->next!=nullptr&&pNode->next->left!=pNode){
               pNode=pNode->next;
           }
           if(pNode->next!=nullptr&&pNode->next->left==pNode)
               res=pNode->next;
       }
        return res;
    }
};
```

58.对称的二叉树

题目描述

请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};
*/
class Solution {
public:
    bool isSymmetrical(TreeNode* pRoot)
    {
        if(pRoot==nullptr)
            return true;
        return cmpSame(pRoot->left,pRoot->right);
    }
    bool cmpSame(TreeNode* l,TreeNode* r)
    {
        if(l==nullptr)
            return r==nullptr;
        if(r==nullptr)
            return false;
        if(l->val!=r->val)
            return false;
        return cmpSame(l->right,r->left)&&cmpSame(l->left,r->right);
    }

};
```

59.按之字形顺序打印二叉树

题目描述

请实现一个函数按照之字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印，其他行以此类推。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};
*/
class Solution {
public:
    vector<vector<int> > Print(TreeNode* pRoot) {
        vector<vector<int>> res;
        if(pRoot==nullptr)
            return res;
        stack<TreeNode*> s1,s2;/*S1保存奇数层的结点，s2保存偶数点的结点*/
        s1.push(pRoot);
        while(!s1.empty()||!s2.empty()){
            vector<int> tmp;
            while(!s1.empty()){/*奇数层，从左到右打印，从右到左保存下一层结点*/
                TreeNode* t=s1.top();
                s1.pop();
                tmp.push_back(t->val);
                if(t->left!=nullptr){
                    s2.push(t->left);

                }
                if(t->right!=nullptr){
                    s2.push(t->right);
                }
            }
            if(tmp.size()>0){
                res.push_back(tmp);
                tmp.clear();
            }

            while(!s2.empty()){/*偶数层，从右到左打印，从左到右保存下一层结点*/
                TreeNode* t=s2.top();
                s2.pop();
                tmp.push_back(t->val);
                if(t->right!=nullptr){
                    s1.push(t->right);
                }
                if(t->left!=nullptr){
                    s1.push(t->left);
                }
            }
            if(tmp.size()>0){
            res.push_back(tmp);
            }
        }
        return res;
```

```
    }

};
```

60.把二叉树打印成多行

题目描述

从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};
*/
class Solution {
public:
        vector<vector<int> > Print(TreeNode* pRoot) {
            vector<vector<int> > res;
            if(pRoot==nullptr)
                return res;
            queue<TreeNode*> q1,q2;
            q1.push(pRoot);
            while(!q1.empty()||!q2.empty()){
                vector<int> t;
                while(!q1.empty()){
                    TreeNode* p=q1.front();
                    q1.pop();
                    t.push_back(p->val);
                    if(p->left!=nullptr)
                        q2.push(p->left);
                    if(p->right!=nullptr)
                        q2.push(p->right);
                }
                if(!t.empty()){
                    res.push_back(t);
                    t.clear();
                }
                while(!q2.empty()){
                    TreeNode* p=q2.front();
                    q2.pop();
                    t.push_back(p->val);
                    if(p->left!=nullptr)
                        q1.push(p->left);
                    if(p->right!=nullptr)
                        q1.push(p->right);
                }
                if(!t.empty())
                    res.push_back(t);
            }
            return res;
        }

};
```

61.序列化二叉树

题目描述

请实现两个函数，分别用来序列化和反序列化二叉树

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};
*/
class Solution {
public:
    char* Serialize(TreeNode *root) {
        vector<int> tmp;
        tmp.clear();
        Seria(root,tmp);
        int* res=new int[tmp.size()];
        for(unsigned long i=0;i<tmp.size();i++){
            res[i]=tmp[i];
        }
        return (char* )res;

    }
    void Seria(TreeNode* root,vector<int>& str)
    {
        if(root==nullptr){
            str.push_back(0x23333);
            return;
        }
        str.push_back(root->val);
        Seria(root->left,str);
        Seria(root->right,str);

    }
    TreeNode* Deserialize(char *str) {
        TreeNode* root=NULL;
        int *p=(int *)str;
        Deseria(root,p);
        return root;
    }
    void Deseria(TreeNode*& root,int*& str)
    {
        if(*str==0x23333){
            ++str;
            return;
        }
            root=new TreeNode(*str);
            Deseria(root->left,++str);
            Deseria(root->right,str);

    }
};
```

62.二叉搜索树的第k个结点

题目描述

给定一棵二叉搜索树，请找出其中的第k小的结点。例如， （5，3，7，2，4，6，8） 中，按结点数值大小顺序第三小结点的值为4。

题解

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};
*/
class Solution {
public:
    TreeNode* KthNode(TreeNode* pRoot, int k)
    {
        if(pRoot==nullptr||k==0)
            return nullptr;
        return GetRes(pRoot,k);
    }

    TreeNode* GetRes(TreeNode* pRoot,int &k)
    {
        TreeNode* res=nullptr;
        if(pRoot->left!=nullptr)
            res=GetRes(pRoot->left,k);
        if(res==nullptr){
            if(k==1)
                return pRoot;
            k--;
        }
        if(res==nullptr&&pRoot->right!=nullptr)
            res=GetRes(pRoot->right,k);
        return res;
    }


};
```

63.数据流中的中位数

题目描述

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。我们使用Insert()方法读取数据流，使用GetMedian()方法获取当前读取数据的中位数。

题解

```
class Solution {
private:
    vector<int> min;
    vector<int> max;
public:
    void Insert(int num)//最小堆的根节点大于最大堆的根节点
    {
         if((min.size()+max.size()&1)==0){/*如果总数目为偶数，分配给最小堆*/
            if(max.size()>0&&num<max[0]){/*如果num小于最大堆的根节点，那么取出最大堆的最大结点
插入最小堆*/
                max.push_back(num);/*加入新数据 先在容器中加入，再调用push_heap()*/
                push_heap(max.begin(),max.end(),less<int>());/*less<int>()为最大堆*/
                num=max[0];
                pop_heap(max.begin(),max.end(),less<int>());/*删除数据  要先调用
pop_heap()，再在容器中删除*/
                max.pop_back();
            }
           min.push_back(num);
            push_heap(min.begin(),min.end(),greater<int>());
        }
        else{/*如果总数目为奇数，分配给最大堆*/
            if(min.size()>0&&num>min[0]){
                min.push_back(num);
                push_heap(min.begin(),min.end(),greater<int>());
                num=min[0];
                pop_heap(min.begin(),min.end(),greater<int>());
                min.pop_back();
            }
            max.push_back(num);
            push_heap(max.begin(),max.end(),less<int>());
        }
    }

    double GetMedian()
    {
        int size=min.size()+max.size();
        if(size==0)
            return 0;
        if((size&1)==1)
           return min[0];
        else{
            return (double)(min[0]+max[0])/2;
        }
        return 0;
    }

};
```

64.滑动窗口的最大值

题目描述

给定一个数组和滑动窗口的大小，找出所有滑动窗口里数值的最大值。例如，如果输入数组{2,3,4,2,6,2,5,1}及滑动窗口的大小3，那么一共存在6个滑动窗口，他们的最大值分别为{4,4,6,6,6,5}； 针对数组{2,3,4,2,6,2,5,1}的滑动窗口有以下6个： {[2,3,4],2,6,2,5,1}， {2,[3,4,2],6,2,5,1}， {2,3,[4,2,6],2,5,1}， {2,3,4,[2,6,2],5,1}， {2,3,4,2,[6,2,5],1}， {2,3,4,2,6,[2,5,1]}。

题解

```
class Solution {
public:
    vector<int> maxInWindows(const vector<int>& num, unsigned int size)
    {
        vector<int> res;
        if(num.empty()||size>num.size()||size==0)
            return res;
        int index=0;
        int max=0;
        int front=0;
        for(int i=0;i<size;i++){
            if(num[i]>max){
                max=num[i];
                front=i;
            }
        }
        res.push_back(max);
        int len=num.size();
        for(unsigned int i=size;i<len;i++){
            if(num[i]>=num[front]){//必须有等于符号，相同最大值要更新front
                res.push_back(num[i]);
                front=i;
            }
            else if(num[i]<num[front]&&i-front<size){
                res.push_back(num[front]);

            }
            else{
                max=0;
                for(int j=front+1;j<=i;j++){
                    if(num[j]>max){
                        max=num[j];
                        front=j;
                    }
                }
                res.push_back(max);
            }
        }
        return res;

    }
};
```

65.矩阵中的路径

目描述

请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。如果一条路径经过了矩阵中的某一个格子，则之后不能再次进入这个格子。 例如 a b c e s f c s a d e e 这样的3 X 4 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。

题解

```
class Solution {
public:
    bool hasPath(char* matrix, int rows, int cols, char* str)
    {
        if(matrix==nullptr||rows<0||cols<0||str==nullptr)
            return false;
        bool * visited=new bool[rows*cols];
        memset(visited,0,rows*cols);
        int length=0;
        for(int col=0;col<cols;col++){
            for(int row=0;row<rows;row++){
                if(hasPathCore(matrix,rows,cols,row,col,str,visited,length)){
                    delete[] visited;
                    return true;
                }

            }
        }
        delete[] visited;
        return false;
    }

    bool hasPathCore(char* matrix,int rows,int cols,int row,int col,char* str,bool*
visited,int length)
    {
        if(str[length]=='\0')
            return true;
        bool hasPath=false;

 if(row>=0&&row<rows&&col>=0&&col<cols&&matrix[row*cols+col]==str[length]&&!visited[ro
w*cols+col]){
            ++length;
            visited[row*cols+col]=true;
            hasPath=hasPathCore(matrix,rows,cols,row+1,col,str,visited,length)
                ||hasPathCore(matrix,rows,cols,row-1,col,str,visited,length)
                ||hasPathCore(matrix,rows,cols,row,col-1,str,visited,length)
                ||hasPathCore(matrix,rows,cols,row,col+1,str,visited,length);
            if(!hasPath){
                length--;
                visited[row*cols+col]=false;
             }
        }
        return hasPath;
    }

};
```

66.机器人的运动范围

题目描述

地上有一个m行和n列的方格。一个机器人从坐标0,0的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，但是不能进入行坐标和列坐标的数位之和大于k的格子。 例如，当k为18时，机器人能够进入方格（35,37），因为3+5+3+7 = 18。但是，它不能进入方格（35,38），因为3+5+3+8 = 19。请问该机器人能够达到多少个格子？

题解

```
class Solution {
public:
    int movingCount(int threshold, int rows, int cols)
    {
        if(threshold<0||rows<0||cols<0)
            return 0;
        bool * visited=new bool[rows*cols];
        memset(visited,0,rows*cols);
        int count=movingCount(threshold,rows,cols,0,0,visited);
        delete[] visited;
        return count;
    }
    int movingCount(int threshold,int rows,int cols,int row,int col,bool* visited)
    {
        int count=0;
        if(check(threshold,rows,cols,row,col,visited)){
            visited[row*cols+col]=true;
            count=1+movingCount(threshold,rows,cols,row+1,col,visited)+
                movingCount(threshold,rows,cols,row-1,col,visited)+
                movingCount(threshold,rows,cols,row,col+1,visited)+
                movingCount(threshold,rows,cols,row,col-1,visited);
        }
        return count;
    }
    bool check(int threshold,int rows,int cols,int row,int col,bool* visited)
    {
        if(threshold>=0&&row<rows&&row>=0&&col>=0&&col<cols&&!visited[row*cols+col]
           &&getNum(row)+getNum(col)<=threshold)
            return true;
        return false;
    }

    int getNum(int number)
    {
        int sum=0;
        while(number>0){
            sum+=number%10;
            number/=10;
        }
        return sum;
    }

};
```