剑指offer

剑指offer题目和代码

数组

二维数组的查找

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。

code

```
数组
```

```
class Solution {
public:
    bool Find(int target, vector<vector<int> > array) {
        int m = array.size();
        if(m == 0){
            return false;
        }
        int n = array[0].size();
        bool found = false;
        int row = 0;
        int col = n - 1; //从左上角开始
        while(row < m && col >= 0){
            if(array[row][col] == target){
                found = true;
                break;
            }
            if(array[row][col] > target){ //当前值大于target，在当前值的左侧继续寻找，其下方和右侧均比当前值更大
                --col;
            }
            else{    //当前值小于target，在当前值的下侧继续寻找，其上方和左侧侧均比当前值更小
                ++row;
            }
        }
        return found;
    }
};
```

网页链接：二维数组的查找

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。

code

数组

```
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
/**
 * 思路：
 * 数组中的数字都在0到n-1的数字范围内。如果数组中没有重复出现的数字，那么当数组排序后数字i就出现在数组中下标为i的元素处。那么数组中如果存在重复数字的话，有些位置的对应的数字就没有出现，而有些位置可能存在多个数字。数组用numbers表示
 那么我们重排这个数组。从第0个元素开始。
 1、比较numbers[i]和i的值，如果i与numbers[i]相等，也就是对数组排序后，numbers[i]就应该在对应的数组的第i个位置处，那么继续判断下一个位置。
 2、如果i和numbers[i]的值不相等，那么判断以numbers[i]为下标的数组元素是什么。
     2.1、如果numbers[numbers[i]]等于numbers[i]的话，那么就是说有两个相同的值了，重复了。找到了重复的数字
     2.2、如果numbers[numbers[i]]不等于numbers[i]的话，那么就将numbers[numbers[i]]和numbers[i]互换。继续进行1的判断。
 3、循环退出的条件是直至数组最后一个元素，仍没有找到重复的数字，数组中不存在重复的数字。
 */
    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;
                }
                swap(numbers[i],numbers[numbers[i]]);
            }
        }
        return false;
    }
};
```

网页链接：数组中重复的数字

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。

code

数组

```
class Solution {
public:
    vector<int> multiply(const vector<int>& A) {
        int len = A.size();
        if(len == 0){
            return vector<int>();
        }
        vector<int> B(len,0);
        B[0] = 1;
        //B[i] = C[i] * D[i]
        //C[0] = 1;
        for(int i = 1; i < len; i++){
            //C[i] = C[i-1] * A[i-1];
            B[i] = B[i-1] * A[i-1];
        }
        //D[len-1] = 1;
        double temp = 1;
        for(int i = len-2; i >= 0; i--){
            //D[i] = D[i+1] * A[i+1];
            temp *= A[i+1];
            B[i] *= temp;
        }
        return B;
    }
};
```

网页链接：构建乘积数组

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两次。请写程序找出这两个只出现一次的数字。

code

知识迁移能力，位运算

```
class Solution {
public:
    void FindNumsAppearOnce(vector<int> data,int* num1,int *num2) {
        int diff = 0;
        int n = data.size();
        for(int i = 0; i < n; i++){
            diff ^= data[i];
        }
        int firstIndex = firstIndexOf1(diff);
        for(int i = 0; i < n; i++){
            if((data[i] >> firstIndex) & 1){
                *num1 ^= data[i];
            }
            else{
                *num2 ^= data[i];
            }
        }
    }
    int firstIndexOf1(int num){
        int index = 0;
        while(num){
            if(num & 1){ //最后一个bit位为1
                break;
            }
            num = num >> 1;
            index++;
        }
        return index;
    }
};
```

网页链接：数组中只出现一次的数字

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

code

知识迁移能力

```
class Solution {
public:
    vector<vector<int> > FindContinuousSequence(int sum) {
        int CurSum = 0;
        int begin = 1;
        int end = 2;
        vector<vector<int> > res;
        if(sum <= 2){
            return res;
        }
        CurSum = begin + end;
        while(begin <= sum / 2){
            if(CurSum == sum){
                vector<int> res_temp;
                GetResult(begin,end,res_temp);
                res.push_back(res_temp);
            }
            while(CurSum > sum && begin <= sum / 2){
                CurSum -=  begin;
                begin++;
                if(CurSum == sum){
                    vector<int> res_temp;
                    GetResult(begin,end,res_temp);
                    res.push_back(res_temp);
                }
            }
            end++;
            CurSum += end;
        }
        return res;
    }
    void GetResult(int start,int end,vector<int>& res){
        for(int i = start; i <= end; i++){
            res.push_back(i);
        }
    }
};
```

网页链接：和为s的连续正数序列

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。

code

知识迁移能力

```
class Solution {
public:
    vector<int> FindNumbersWithSum(vector<int> array,int sum) {
        vector<int> res;
        if(array.size() == 0){
            return res;
        }
        int n = array.size();
        int left = 0;
        int right = n - 1;
        int left_res = -1;
        int right_res = -1;
        int minMul = INT_MAX;
        while(left < right){
            if(array[left] + array[right] == sum){
                if(array[left] * array[right] < minMul){
                    minMul = array[left] * array[right];
                    left_res = left;
                    right_res = right;
                }
                --right;
                ++left;
            }
            if(array[left] + array[right] > sum){
                --right;
            }
            else{
                ++left;
            }
        }
        if(left_res == -1 && right_res == -1){
            return res;
        }
        res.push_back(array[left_res]);
        res.push_back(array[right_res]);
        return res;
```

```
    }
};
```

网页链接：和为s的两个数字

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。

code

查找和排序

```
class Solution {
public:
    int minNumberInRotateArray(vector<int> rotateArray) {
        int len = rotateArray.size();
        if(len==0){
            return 0;
        }
        if(len == 1){
            return rotateArray[0];
        }
        int left = 0;
        int right = len - 1;
        int mid = 0;
        if(rotateArray[left] < rotateArray[right]){
            return rotateArray[0];
        }
        while(right - left > 1){
            mid = left + (right - left) / 2;
            //特殊情况，如【1,0，1,1,1】，不能使用二分法，只能顺序查找
            if(rotateArray[mid] == rotateArray[left] && rotateArray[mid] ==
rotateArray[right]){
                return findMin(rotateArray,left,right);
            }
            //二分法查找
            else if(rotateArray[mid] >= rotateArray[left]){
                left = mid;
            }
            else{
                right = mid;
            }
        }
```

```
            return rotateArray[right];
        }
        int findMin(vector<int> rotateArray,int left,int right){
            int min = INT_MAX;
            for(int i = left; i <= right; i++){
                if(rotateArray[i] < min){
                    min = rotateArray[i];
                }
            }
            return min;
        }
    };
```

网页链接：旋转数组的最小数字

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。

code

时间效率，数组

```
class Solution {
public:
    int MoreThanHalfNum_Solution(vector<int> numbers) {
        int n = numbers.size();
        if( n == 0 ){
            return 0;
        }
        if( n == 1 ){
            return numbers[0];
        }
        int res = numbers[0];
        int count = 1;
        //确定哪一个数可能出现的次数大于数组长度的一半，不完全正确
        for(int i = 1; i < n; i++){
            if(numbers[i] == res){
                count++;
            }
            else{
                count--;
                if(count == 0){
                    res = numbers[i];
```

```
                    count = 1;
                }
            }
        }
        //对可能的数字进行确认
        int cnt = 0;
        for(int i = 0; i < n; i++){
            if(numbers[i] == res){
                cnt++;
            }
        }
        if(cnt > n/2){
            return res;
        }
        return 0;
    }
};
```

网页链接：数组中出现次数超过一半的数字

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。

code

时间效率，优先级队列，数组

```
class Solution {
public:
    vector<int> GetLeastNumbers_Solution(vector<int> input, int k) {
        vector<int> res;
        int n = input.size();
        if(k <= 0 || k > n){
            return res;
        }
        priority_queue<int, vector<int>, less<int> > pq; //维护一个最大堆（优先级队列，数字大的在前面，先出队列）
        for(int i = 0; i < n ;i++){
            if(pq.size() < k){
                pq.push(input[i]);
            }
            else{
                if(input[i] < pq.top()){
                    pq.pop();
```

```
                    pq.push(input[i]);
                }
            }
        }
        //pq内存放的为k个最小值，他们是按照从大到小的顺序排列
        while(!pq.empty()){
            res.push_back(pq.top());
            pq.pop();
        }
        return res;
    }
};
```

网页链接：最小的k个数

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)

code

时间效率，数组

```
class Solution {
public:
    int FindGreatestSumOfSubArray(vector<int> array) {
        int n = array.size();
        if(n == 0){
            return 0;
        }
        //动态规划
        vector<int> dp(n,0); //dp[i]表示最后一个数为array[i]的子数组的最大和
        dp[0] = array[0];
        for(int i = 1; i < n; i++){
            if(dp[i-1] < 0){
                dp[i] = array[i];
            }
            else{
                dp[i] = dp[i-1]+array[i];
            }
        }
        int max = INT_MIN;
```

```
        for(int i = 0; i<n;i++){
            if(dp[i]>max){
                max = dp[i];
            }
        }
        return max;
    }
};
```

网页链接：连续子数组的最大和

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。

code

时间效率,字符串

```
class Solution {
public:
    string PrintMinNumber(vector<int> numbers) {
        int n = numbers.size();
        if(n == 0){
            return "";
        }
        if(n == 1){
            stringstream ss;
            string str;
            ss<<numbers[0];
            ss>>str;
            return str;
        }
        string res;
        string* input = new string[n];
        for(int i = 0 ; i < n; i++){
            stringstream ss;
            ss<<numbers[i];
            ss>>input[i];
        }
        sort(input,input+n,compare);
        for(int i = 0; i < n ;i++){
            res += input[i];
        }
        delete[] input;
```

```
            return res;
        }
        static bool compare(const string& str1,const string& str2){
            string s1 = str1 + str2;
            string s2 = str2 + str1;
            return s1 < s2;
        }
    };
```

网页链接：把数组排成最小的数

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

code

时间空间效率的平衡

```
class Solution {
public:
    int InversePairs(vector<int> data) {
        if(data.size() <= 1){
            return 0;
        }
        int n = data.size();
        vector<int> copy(data);
        long long int cnt = InversePairsCore(data,copy,0,n-1);
        return cnt % 1000000007;
    }
    long long int InversePairsCore(vector<int>& data,vector<int>& copy,int
start,int end){
        if(start == end){
            copy[start] = data[start];
```

```
            return 0;
        }
        int mid = start + (end - start) / 2;
        long long int leftCnt = InversePairsCore(copy,data,start,mid);
        long long int rightCnt = InversePairsCore(copy,data,mid+1,end);
        int CopyIndex = end;
        int i = mid; //i指向前半个数组的最后
        int j = end; //j指向后半个数组的最后
        long long int mergeCnt = 0;
        while( i >= start && j >= mid + 1){
            if(data[i] > data[j]){
                copy[CopyIndex--] = data[i--];
                mergeCnt += j - mid;
            }
            else{
                copy[CopyIndex--] = data[j--];
            }
        }
        while(i >= start){
            copy[CopyIndex--] = data[i--];
        }
        while(j >= mid + 1){
            copy[CopyIndex--] = data[j--];
        }
        return leftCnt + rightCnt + mergeCnt;
    }
};
```

网页链接：数组中的逆序对

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。

code

知识迁移能力

```
class Solution {
public:
    int GetNumberOfK(vector<int> data ,int k) {
        int n = data.size();
        if(n == 0){
            return 0;
        }
        int left = GetfirstIndexOfK(data,k);
```

```
        int right =GetlsatIndexOfK(data,k);
        if(left > -1 && right > -1)
            return right - left + 1;
        else
            return 0;
    }
    int GetfirstIndexOfK(vector<int> data ,int k){
        int n = data.size();
        if(n == 0){
            return 0;
        }
        int left = 0;
        int right = n - 1;
        while(left <= right){
            int mid = left + ( right - left ) / 2;
            if(data[mid] == k){
                if((mid - 1 >= 0) && data[mid - 1]!=k || mid == 0){
                    return mid;
                }
                else{
                    right = mid - 1;
                }
            }
            else if(data[mid] < k){
                left = mid + 1;
            }
            else{
                right = mid - 1;
            }
        }
        return -1;
    }
    int GetlsatIndexOfK(vector<int> data ,int k){
        int n = data.size();
        if(n == 0){
            return 0;
        }
        int left = 0;
        int right = n - 1;
        while(left <= right){
            int mid = left + ( right - left ) / 2;
            if(data[mid] == k){
                if((mid + 1 <=  n-1) && data[mid + 1]!=k || mid == n-1){
                    return mid;
                }
                else{
                    left = mid + 1;
                }
            }
            else if(data[mid] < k){
                left = mid + 1;
            }
            else{
                right = mid - 1;
```

```
            }
        }
        return -1;
    }
};
```

网页链接：数字在排序数组中出现的次数

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]}。

code

栈和队列

```
class Solution {
public:
    vector<int> maxInWindows(const vector<int>& num, unsigned int size)
    {
        vector<int> res;
        int n = num.size();
        if(n < size || n <= 0){return res;}
        if(size == 1){
            return num;
        }
        deque<int> q; //存储的应该是下标，不应该是值。队列第一个数据存放的为当前滑动窗口的最大值的索引
        for(int i =  0; i < n; i++){
            if(q.empty()){ //队列为空
                q.push_back(i);
            }
            else{
                if(num[i] > num[q.front()] && i - q.front() < size){ //新进入滑动窗口的数据比队列中的最大值大，并且滑动窗口的宽度没有超过阈值
                    while(!q.empty()){
                        q.pop_front();
                    }
                    q.push_back(i);
                }
                else{
                    if(i - q.front() >= size){ //窗口大小大于设定值
```

```
                        q.pop_front();
                    }
                    while(num[i] > num[q.back()] && !q.empty()){ //更新队列
中的值，确保队头为最大值索引
                        q.pop_back();
                    }
                    q.push_back(i);
                }
            }
            if(i >= size - 1){
                res.push_back(num[q.front()]);
            }
        }
        return res;
    }
};
```

网页链接：滑动窗口的最大值

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。

code

代码的完整性，数组

```
class Solution {
public:
    void reOrderArray(vector<int> &array) {
        //双指针法不可用，结果会改变数据的相对次序，与题意不符
        int len = array.size();
        if(len <= 1){
            return;
        }
        vector<int> odd; //奇数
        vector<int> even;  //偶数
        for(int i = 0; i < len; i++){
            if(array[i]& 1){
                odd.push_back(array[i]);
            }
            else{
                even.push_back(array[i]);
            }
        }
```

```
        int len1 = odd.size();
        int len2 = even.size();
        for(int i = 0; i < len1 + len2; i++){
            if(i < len1){
                array[i] = odd[i];
            }
            else{
                array[i] = even[i - len1];
            }
        }
    }
};
```

网页链接：数值的整数次方调整数组顺序使奇数位于偶数前面

注：链接内为剑指offer原书内的题目解法，与本题有一些差异。剑指offer中不要求变化后相对位置不变，故可用双指针法解法。

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。

code

优先级队列

```
class Solution {
public:
    void Insert(int num)
    {
        if(size&1){ //数据流中已有奇数个数据,数据存放在最小堆
            if(num < maxHeap.top()){
                int temp = maxHeap.top();
                maxHeap.pop();
                maxHeap.push(num);
                minHeap.push(temp);
            }
            else{
                minHeap.push(num);
            }
        }
        else{//数据流中已有偶数个数据,数据存放在最大堆
            if(!minHeap.empty() && num > minHeap.top()){
                int temp = minHeap.top();
```

```
                minHeap.pop();
                minHeap.push(num);
                maxHeap.push(temp);
            }
            else{
                maxHeap.push(num);
            }
        }
        size++;
    }

    double GetMedian()
    {
        if(size&1){
            return double(maxHeap.top());
        }
        else{
            return double(maxHeap.top() + minHeap.top()) / 2.0;
        }
    }
    //维护一个最大堆和最小堆
    priority_queue<int, vector<int>, less<int> > maxHeap; //最大堆
    priority_queue<int, vector<int>, greater<int> > minHeap; //最小堆
    int size = 0;
};
```

网页链接：数据流中的中位数

扑克牌顺子

题目描述

> 从扑克牌中随机抽 5 张牌，判断是不是一个顺子， 即这 5 张牌是不是连续的。2～10 为数字本身， A 为 1。 J 为 11、Q 为 12、 为 13。小王可以看成任意数字。。

code

抽象思维能力

```
class Solution {
public:
    bool IsContinuous( vector<int> numbers ) {
        if(numbers.size() <= 0){
            return false;
        }
        if(numbers.size() == 1){
            return true;
        }
```

```
        int len = numbers.size();
        sort(numbers.begin(),numbers.end(),compare);
        int numOfZero = 0;
        int numOfGap = 0;
        for(int i = 0; i < len && numbers[i] == 0; i++){
            numOfZero++;
        }
        int cur = numOfZero;
        while(cur < len - 1){
            if(numbers[cur] == numbers[cur+1]){
                return false;
            }
            else{
                numOfGap = numOfGap + numbers[cur + 1] - numbers[cur] - 1;
            }
            ++cur;
        }
        if(numOfZero >= numOfGap){
            return true;
        }
        return false;
    }
    static bool compare(const int a,const int b){
        return a < b;
    }
};
```

网页链接：扑克牌顺子

字符串

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。

code

字符串

```
class Solution {
public:
        void replaceSpace(char *str,int length) { //
        if(str == nullptr || length < 0){
```

```
            return;
        }
        int spaceNum = findSpaceNum(str);
        int newlength = length + 2*spaceNum;
        char* src = (char*)malloc(length*sizeof(char));
        memcpy(src,str,length);
        src = (char*)realloc(src,newlength*sizeof(char));
        int count = spaceNum;
        for(int i = length - 1; i >= 0; i--){
            if(src[i]==' '){
                str[i+2*count] = '0';
                str[i+2*count-1] = '2';
                str[i+2*count-2] = '%';
                count--;
            }
            else{
                str[i+2*count] = src[i];
            }
        }
        }
    int findSpaceNum(char *str){
        int length = strlen(str);
        int num = 0;
        for(int i = 0; i < length; i++){
            if(str[i] == ' '){
                num++;
            }
        }
        return num;
    }
};
```

网页链接：替换空格

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。

code

分解让复杂问题简单化，字符串

```
class Solution {
public:
    vector<string> Permutation(string str) {
        vector<string> res;
        if(str.empty()){
            return res;
        }
        //可能会有重复的排列出现，需要排除
        set<string> res_temp;
        PermutationCore(res_temp,str,0);

        for(auto iter = res_temp.begin(); iter!=res_temp.end();++iter){
            res.push_back(*iter);
        }
        return res;
    }
    void PermutationCore(set<string>& result, string str,int pos){
        int n = str.size();
        if(pos == n){
            result.insert(str);
            return;
        }
        for(int i = pos; i < n; i++){
            swap(str[i],str[pos]);
            PermutationCore(result,str,pos+1);
            swap(str[i],str[pos]);
        }
    }
};
```

网页链接：字符串的排列

注： 链接内的方法没有考虑去重，需要加上去重操作（可利用set的特性进行去重）

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）

code

时间空间效率的平衡,哈希表

```
class Solution {
public:
    int FirstNotRepeatingChar(string str) {
        if(str.size() == 0){
            return -1;
        }
        int len = str.size();
        int hashTable[256] = {0};
        for(int i = 0; i < len; i++){
            hashTable[str[i] - '\0']++;
        }
        for(int i = 0; i < len; i++){
            if(hashTable[str[i] - '\0'] == 1){
                return i;
            }
        }
        return -1;
    }
};
```

网页链接：第一个只出现一次的字符

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。

code

字符串

```
class Solution
{
public:
    Solution():index(0){
        for(int i = 0; i<256;i++){
            table[i] = -1;
        }
```

```
    }
  //Insert one char from stringstream
    void Insert(char ch)
    {
        if(table[ch - '\0'] == -1){ //-1表示已经该字符还没有出现过
            table[ch - '\0'] = index;
        }
        else if(table[ch - '\0'] >= 0){
            table[ch - '\0'] = -2; //-2表示已经该字符已经出现2次及以上
        }
        index++;
    }
  //return the first appearence once char in current stringstream
    char FirstAppearingOnce()
    {
        int minIndex = 256;
        char res;
        for(int i = 0; i<256; i++){
            if(table[i] >= 0 && table[i] < minIndex){
                res = char(i);
                minIndex = table[i];
            }
        }
        if(minIndex == 256){
            return '#';
        }
        return res;
    }
    char table[256];
    int index;
};
```

网页链接：字符流中第一个不重复的字符

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！

code

知识迁移能力

```
class Solution {
public:
```

```
    string LeftRotateString(string str, int n) {
        if(str.size() == 0 || n % str.size() == 0){
            return str;
        }
        int len = str.size();
        n = n % len;
        /*
        //直观的解决方法，采用辅助空间
        char temp[n];
        for(int i = 0; i < n; i++){
            temp[i] = str[i];
        }
        for(int i = 0; i < len; i++){
            if(i < len - n){
                str[i] = str[i+n];
            }
            else{
                str[i] = temp[i - (len - n)];
            }
        }
        return str;
        */
        //只用一个固定的辅助空间，通过多次的反转字符串实现
        Reserve(str,0,n-1);
        Reserve(str,n,len-1);
        Reserve(str,0,len-1);
        return str;
    }
    void Reserve(string& str, int start, int end){
        int left = start;
        int right = end;
        while(left < right){
            char temp = str[left];
            str[left] = str[right];
            str[right] = temp;

            left++;
            right--;
        }
    }
};
```

网页链接：左旋转字符串

翻转单词顺序列

题目描述

牛客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？

code

知识迁移能力

```
class Solution {
public:
    string ReverseSentence(string str) {
        if(str.size() <= 1){
            return str;
        }
        int len = str.size();
        Reserve(str,0,len-1);
        int start = 0;
        for(int i = 0; i <= len; i++){
            if(str[i] == ' ' || str[i] == '\0'){
                Reserve(str,start,i-1);
                start = i + 1;
            }
        }
        return str;
    }
    void Reserve(string& str, int start, int end){
        int l = start;
        int r = end;
        while(l < r){
            char temp = str[l];
            str[l] = str[r];
            str[r] = temp;
            l++;
            r--;
        }
    }
};
```

网页链接:翻转单词顺序列

把字符串转化为数字

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
```

code

综合

```
class Solution {
public:
    int StrToInt(string str) {
        if(str.size() == 0){
            return 0;
        }
        bool IsPositive = true;
        int cur = 0;
        if(str[0] == '-'){
            IsPositive = false;
            cur++;
        }
        if(str[0] == '+'){
            cur++;
        }
        int res = 0;
        int carry = 1;
        for(int i = str.size() - 1; i >= cur; i--){
            if(str[i] < '0' || str[i] > '9'){
                return 0;
            }
            else{
                res += (str[i] - '0') * carry;
                carry *= 10;
            }
```

```
            if((IsPositive && res > 0x7FFFFFFF) || (!IsPositive && res >
0x80000000) ){
                return 0;
            }
        }
        return IsPositive ? res: -res;
    }
};
```

网页链接：把字符串转化为数字

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

code

字符串，动态规划

```
class Solution {
public:
    bool match(char* str, char* pattern)
    {
         //动态规划
        // 1, If p.charAt(j) == s.charAt(i) :  dp[i][j] = dp[i-1][j-1];
        // 2, If p.charAt(j) == '.' : dp[i][j] = dp[i-1][j-1];
        // 3, If p.charAt(j) == '*':
        // here are two sub conditions:
        //        1   if p.charAt(j-1) != s.charAt(i) : dp[i][j] = dp[i][j-
2]  //in this case, a* only counts as empty
        //        2   if p.charAt(j-1) == s.charAt(i) or p.charAt(j-1) ==
'.':
        //                       dp[i][j] = dp[i-1][j]    //in this case,
a* counts as multiple a
        //                   or dp[i][j] = dp[i][j-1]   // in this case,
a* counts as single a
        //                   or dp[i][j] = dp[i][j-2]   // in this case,
a* counts as empty
        int m = strlen(str);
        int n = strlen(pattern);
        //dp[i][j]表示str[0,i)与pattern[0,j)是否匹配
        vector<vector<bool> > dp(m+1,vector<bool>(n+1,false));
        dp[0][0] = true;
        for(int j = 1; j <= n; j++){
```

```
            if(pattern[j - 1] == '*'){
                dp[0][j] = dp[0][j-2];
            }
        }
        for(int i = 1; i <= m;i++){
            for(int j= 1; j <= n; j++){
                if(str[i-1] == pattern[j-1] || pattern[j-1]=='.'){
                    dp[i][j] = dp[i-1][j-1];
                }
                else if(pattern[j-1]=='*'){
                    if(str[i-1] != pattern[j-2]){
                        dp[i][j] = dp[i][j-2];
                    }
                    if(str[i-1] == pattern[j-2] || pattern[j-2]=='.'){
                        dp[i][j] = dp[i][j-1] || dp[i-1][j] || dp[i][j-2];
                    }
                }
            }
        }
        return dp[m][n];
    }
```

网页链接：正则表达式匹配

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。

code

字符串

```
class Solution {
public:
    bool isNumeric(char* string)
    {
        //A[.[B]][e|EC] 或 .B[e|EC]
        //A,C是+，-开头的数字(可以没有+，-)，B是数字，且不能有+，-
        if(string == nullptr){
            return false;
        }
        bool isNum = ScanInter(&string);
        //如果出现.,接下来是小数部分
        if(*string == '.'){
```

```
            ++string;
            //小数可以没有整数部分，如.123
            //小数后面可以没有数字，如123.
            //小数的前面和后面都可以有数字
            isNum = ScanUnsignInter(&string) || isNum;
        }
        //如果出现e|E,接下来是指数部分
        if(*string == 'e' || *string == 'E'){
            ++string;
            //指数前后必须都有数字
            isNum = isNum && ScanInter(&string);
        }
        return isNum && *string == '\0';
    }
    bool ScanInter(char** string){
        if(**string == '+' || **string == '-'){
            ++(*string);
        }
        return ScanUnsignInter(string);
    }
    bool ScanUnsignInter(char** string){
        const char* src = *string;
        while(**string != '\0' && **string <= '9' && **string >= '0'){
            ++(*string);
        }
        return *string > src;
    }
};
```

网页链接：表示数值的字符串

链表

从头到尾打印链表

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

code

链表

```
/**
*  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;
        }
        //使用递归
        //help_recursively(head,res);
        //不使用递归,使用栈（先入后出）
        stack<int> s;
        while(head){
            s.push(head->val);
            head = head->next;
        }
        while(!s.empty()){
            res.push_back(s.top());
            s.pop();
        }
        return res;
    }
    void help_recursively(ListNode* head,vector<int>& res){
        if(head){
            if(head->next){
                help_recursively(head->next,res);
            }
            res.push_back(head->val);
        }
    }
};
```

网页链接：从头到尾打印链表

链表中倒数第k个节点

题目描述

> 输入一个链表，输出该链表中倒数第k个结点。

code

代码的鲁棒性，链表

```
/*
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 == nullptr || k <= 0){
            return nullptr;
        }
        ListNode* pre = pListHead; //先前进k步
        ListNode* cur = pListHead; //始终与pre相差k个节点，当pre到达链表尾后，
cur即为需返回的节点
        for(int i = 0; i < k; i++){
            if(pre == nullptr){
                return nullptr;
            }
            else{
                pre = pre->next;
            }
        }
        while(pre){
            pre = pre->next;
            cur = cur->next;
        }
        return cur;
    }
};
```

网页链接：链表中倒数第k个节点

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。

code

代码的鲁棒性，链表

```
/*
struct ListNode {
```

```
        int val;
        struct ListNode *next;
        ListNode(int x) :
                        val(x), next(NULL) {
        }
};*/
class Solution {
public:
    ListNode* ReverseList(ListNode* pHead) {
        if(pHead == nullptr){
            return pHead;
        }
        ListNode* pReversehead = nullptr;
        ListNode* pre = nullptr;
        ListNode* cur = pHead;
        while(cur){
            ListNode* pNext = cur->next;
            if(pNext == nullptr){
                pReversehead = cur;
            }
            cur->next = pre;
            pre = cur;
            cur = pNext;
        }
        return pReversehead;
    }
};
```

网页链接：反转链表

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。

code

代码的鲁棒性，链表

```
/*
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 == nullptr){
            return pHead2;
        }
        if(pHead2 == nullptr){
            return pHead1;
        }
        /*
        //递归
        if(pHead1->val > pHead2->val){
            pHead2->next = Merge(pHead1,pHead2->next);
            return pHead2;
        }
        else{
            pHead1->next = Merge(pHead1->next,pHead2);
            return pHead1;
        }
        */
        //非递归
        ListNode* temp = new ListNode(-1);
        ListNode* res = temp;
        while(pHead1 && pHead2){
            if(pHead1->val > pHead2->val){
                temp->next = pHead2;
                pHead2 = pHead2->next;
            }
            else{
                temp->next = pHead1;
                pHead1 = pHead1->next;
            }
            temp = temp->next;
        }
        if(pHead1){
            temp->next = pHead1;
        }
        else{
            temp->next = pHead2;
        }
        return res->next;
    }
};
```

网页链接：合并两个排序的链表

链表中环的入口节点

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。

code

链表

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
        val(x), next(NULL) {
    }
};
*/
class Solution {
public:
    ListNode* EntryNodeOfLoop(ListNode* pHead)
    {
        ListNode* meet = meetingNode(pHead);
        if(meet == nullptr){
            return nullptr;
        }
        int CircleLen = 1;
        ListNode* cur = meet;
        while(cur->next != meet){
            cur = cur->next;
            ++CircleLen;
        }
        ListNode* slow = pHead;
        ListNode* fast = pHead;
        for(int i = 0; i < CircleLen; i++){
            fast = fast->next;
        }
        while(slow != fast){
            slow = slow->next;
            fast = fast->next;
        }
        return fast;
    }
    ListNode* meetingNode(ListNode* pHead){
        if(pHead == nullptr){
            return nullptr;
        }
        ListNode* slow = pHead->next;
        if(slow == nullptr){
            return nullptr;
        }
        ListNode* fast = slow->next;
        while(slow && fast){
```

```
            if(fast == slow){
                return fast;
            }
            slow = slow->next;
            fast = fast->next;
            if(fast){
                fast = fast->next;
            }
        }
        return nullptr;
    }
};
```

网页链接:链表中环的入口节点

删除链表中的重复节点

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

code

链表

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
        val(x), next(NULL) {
    }
};
*/
class Solution {
public:
    ListNode* deleteDuplication(ListNode* pHead)
    {
        if(pHead == nullptr){
            return pHead;
        }
        ListNode* pre = nullptr;
        ListNode* cur = pHead;
        while(cur){
            ListNode* pNext = cur->next;
            bool needDel = false;
            if(pNext && cur->val == pNext->val){
```

```
                needDel = true;
            }
            if(!needDel){ //没有重复节点
                pre = cur;
                cur = cur->next;
            }
            else{//存在重复节点
                ListNode* pToBeDel = cur;
                int val = cur->val;
                while(pToBeDel && pToBeDel->val == val){
                    pNext = pToBeDel->next;
                    delete pToBeDel;
                    pToBeDel = nullptr;
                    pToBeDel = pNext;
                }
                if(pre == nullptr){
                    pHead = pNext;
                }
                else{
                    pre->next = pNext;
                }
                cur = pNext;
            }
        }
        return pHead;
    }
};
```

网页链接：删除链表中的重复节点

两个链表的第一个公共节点

题目描述

> 输入两个链表，找出它们的第一个公共结点。

code

时间空间效率的平衡

```
/*
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 == nullptr || pHead2 == nullptr){
            return nullptr;
        }
        int l1 = GetListLength(pHead1);
        int l2 = GetListLength(pHead2);
        ListNode* longList = pHead1;
        ListNode* shortList = pHead2;
        int diff = l1 - l2;
        if(l1 < l2){
            diff = l2 -l1;
            longList = pHead2;
            shortList = pHead1;
        }
        for(int i = 0; i < diff; i++){
            longList = longList->next;
        }
        while(shortList && longList && shortList != longList){
            shortList = shortList->next;
            longList = longList->next;
        }
        return shortList;
    }
    int GetListLength(ListNode* pHead){
        int len = 0;
        while(pHead){
            len++;
            pHead = pHead->next;
        }
        return len;
    }
};
```

网页链接：两个链表的第一个公共节点

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）

code

分解让复杂问题简单，链表

```
/*
struct RandomListNode {
    int label;
    struct RandomListNode *next, *random;
    RandomListNode(int x) :
            label(x), next(NULL), random(NULL) {
    }
};
*/
class Solution {
public:
    RandomListNode* Clone(RandomListNode* pHead)
    {
        if(pHead == nullptr){
            return nullptr;
        }
        CloneNode(pHead);
        CloneRandom(pHead);
        RandomListNode* res = DividApart(pHead);
        return res;
    }
    //仅拷贝节点和其下一个节点，不对任意指针进行处理
    void CloneNode(RandomListNode* pHead){
        RandomListNode* cur = pHead;
        while(cur){
            RandomListNode* temp = new RandomListNode(cur->label); //这里需要新建一个节点，不能直接将指针复制（clone节点，需要多出一个节点，而不是对弄一个节点的引用）
            temp->next = cur->next;
            temp->random = nullptr;
            cur->next = temp;
            cur = temp->next;
        }
    }
    //节点拷贝完成后，对节点的任意指针进行处理
    void CloneRandom(RandomListNode* pHead){
        RandomListNode* cur = pHead;
        while(cur){
            RandomListNode* temp = cur->next;
            if(cur->random){
                temp->random = cur->random->next;
            }
            cur = temp->next;
        }
    }
    //节点按照奇偶分离
    RandomListNode* DividApart(RandomListNode* pHead){
        RandomListNode* cur = pHead;
        RandomListNode* Cloned = nullptr;
        RandomListNode* res = nullptr;
        if(cur){
            res = Cloned = cur->next;
            cur->next = Cloned->next;
```

```
            cur = cur->next;
        }
        while(cur){
            Cloned->next = cur->next;
            Cloned = Cloned->next;
            cur->next = Cloned->next;
            cur = cur->next;
        }
        return res;
    }
};
```

网页链接：复杂链表的复制

树

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。

code

树

```
/**
 * 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 || vin.size() != pre.size()){
            return nullptr;
        }
        int length = pre.size();
        return reConstructBinaryTreeCore(pre,vin,0,length-1,0,length-1);
    }
```

```
    TreeNode* reConstructBinaryTreeCore(vector<int> pre,vector<int> vin,int
preStart,int preEnd,int vinStart,int vinEnd){
        int rootVal = pre[preStart]; //前序遍历的第一个节点即为根节点
        TreeNode* root = new TreeNode(rootVal);
        //递归终止条件
        if(preStart == preEnd){
            if(vinStart == vinEnd && pre[preStart] == vin[vinStart]){
                return root;
            }
            else{
                //throw std::exception("invalid input");
                return nullptr;
            }
        }
        int rootVinIndex = vinStart;
        //寻找根节点在中序遍历中的位置，根节点前的节点为左子树，之后的节点为右子树
        while(rootVinIndex <= vinEnd && vin[rootVinIndex]!= rootVal){
            rootVinIndex++;
        }
        //递归构造左右子树
        int leftLength = rootVinIndex - vinStart + 1;
        if(leftLength){
            root->left = reConstructBinaryTreeCore(pre,vin,preStart +
1,preStart + leftLength,vinStart,vinStart + rootVinIndex - 1);
        }
        if(rootVinIndex < vinEnd){
            root->right = reConstructBinaryTreeCore(pre,vin,preStart +
leftLength,preEnd,vinStart+ leftLength,vinEnd);
        }
        return root;
    }
};
```

网页链接：重建二叉树

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）

code

代码的鲁棒性,树

```
/*
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 res = false;
        if(pRoot1 && pRoot2){
            if(pRoot1->val == pRoot2->val){
                res = DoesTree1hasTree2(pRoot1,pRoot2);
            }
            if(!res){
                res = HasSubtree(pRoot1->left,pRoot2); //递归判断左子树
            }
            if(!res){
                res = HasSubtree(pRoot1->right,pRoot2); //递归判断右子树
            }
        }
        return res;
    }
    bool DoesTree1hasTree2(TreeNode* pRoot1, TreeNode* pRoot2){
        if(pRoot2 == nullptr){//先判断子树是否为空树，后判断要匹配的树，顺序不能颠
倒
            return true;
        }
        if(pRoot1 == nullptr){
            return false;
        }
        if(pRoot1->val != pRoot2->val){
            return false;
        }
        return DoesTree1hasTree2(pRoot1->left,pRoot2->left) &&
DoesTree1hasTree2(pRoot1->right,pRoot2->right);
    }
};
```

网页链接：树的子结构

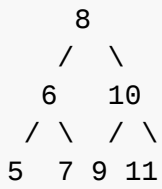
二叉树的镜像

题目描述

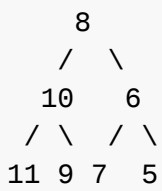
操作给定的二叉树，将其变换为源二叉树的镜像。

输入描述:
二叉树的镜像定义：

源二叉树



镜像二叉树



code

面试思路，树

```
/*
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 == nullptr){
            return;
        }
        if(!pRoot->left && !pRoot->right){
            return;
        }
        //交换左右节点
        TreeNode* temp = pRoot->left;
        pRoot->left = pRoot->right;
        pRoot->right = temp;

        //如果左右子树存在，递归镜像左右子树
        if(pRoot->left){
            Mirror(pRoot->left);
        }
        if(pRoot->right){
            Mirror(pRoot->right);
        }
    }
};
```

网页链接：二叉树的镜像

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。

code

树

```
/*
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 isSymmetricalCore(pRoot,pRoot);
    }
    bool isSymmetricalCore(TreeNode* pRoot1,TreeNode* pRoot2){
        if(pRoot1 == nullptr && pRoot2 == nullptr){
            return true;
        }
        if(pRoot1 == nullptr || pRoot2 == nullptr){
            return false;
        }
        if(pRoot1->val != pRoot2->val){
            return false;
        }
        return isSymmetricalCore(pRoot1->left,pRoot2->right) &&
isSymmetricalCore(pRoot1->right,pRoot2->left);
    }
};
```

网页链接:对称的二叉树

二叉树的下一个节点

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。

code

树

```
/*
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* pNext = nullptr;
        //如果一个节点有右子树，那么它的下一个节点就是它的右子树的最左子节点
        if(pNode->right){
            TreeLinkNode* Right = pNode->right;
            while(Right->left){
                Right = Right->left;
            }
            pNext = Right;
        }
        else if(pNode->next){
            TreeLinkNode* cur = pNode;
            TreeLinkNode* parent = pNode->next;
            //如果节点是它父节点的左子节点，那么它的下一个节点就是它的父节点
            //如果节点既没有右子树，而且父节点的右子节点：
            //    沿着父节点的指针一路向上遍历，直到找到一个是它父节点的左子节点的节点。此时其父节点就是需要找的下一个节点
            while(parent && cur == parent->right){
                 cur = parent;
                 parent = parent->next;
            }
```

```
            pNext = parent;
        }
        return pNext;
    }
};
```

网页链接：二叉树的下一个节点

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。

code

举例让抽象具体化，树，队列

```
/*
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 == nullptr){
            return res;
        }
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()){
            TreeNode* temp = q.front();
            res.push_back(temp->val);
            q.pop();
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
        }
        return res;
```

```
    }
};
```

网页链接：从上往下打印二叉树

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。

code

树

```
/*
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) {
            if(pRoot == nullptr){
                return vector<vector<int> >();
            }
            queue<TreeNode*> q;
            int levelCnt = 1;
            int cnt = 0;
            q.push(pRoot);
            vector<vector<int> > res;
            vector<int> res_temp;
            while(!q.empty()){
                TreeNode* temp = q.front();
                q.pop();
                ++cnt;
                res_temp.push_back(temp->val);
                if(temp->left){
                    q.push(temp->left);
                }
                if(temp->right){
                    q.push(temp->right);
                }
```

```
                if(cnt == levelCnt){
                    res.push_back(res_temp);
                    res_temp.clear();
                    cnt = 0;
                    levelCnt = q.size();
                }
            }
            return res;
        }
};
```

网页链接：把二叉树打印成多行

之字型顺序打印二叉树

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。

code

树

```
/*
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) {
        if(pRoot == nullptr){
            return vector<vector<int> >();
        }
        vector<vector<int> > res;
        stack<TreeNode*> s[2];
        int cur = 0;
        int next = 1;
        s[cur].push(pRoot);
        vector<int> res_temp;
        while(!s[0].empty() || !s[1].empty()){
            TreeNode* temp = s[cur].top();
```

```
            s[cur].pop();
            res_temp.push_back(temp->val);
            if(cur == 0){
                if(temp->left){
                    s[next].push(temp->left);
                }
                if(temp->right){
                    s[next].push(temp->right);
                }
            }
            else{
                if(temp->right){
                    s[next].push(temp->right);
                }
                if(temp->left){
                    s[next].push(temp->left);
                }
            }
            if(s[cur].empty()){
                res.push_back(res_temp);
                res_temp.clear();
                cur = 1 - cur;
                next = 1 - next;
            }
        }
        return res;
    }
};
```

网页链接：之字型顺序打印二叉树

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)

code

举例让抽象具体化，树

```
/*
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_temp;
        vector<vector<int> > res;
        if(root == nullptr){
            return res;
        }
        int curSum = 0;
        FindPathCore(res, res_temp, root,curSum,expectNumber);
        return res;
    }
    void FindPathCore(vector<vector<int> >& res, vector<int>& res_temp,
TreeNode* root, int curSum, int expectNumber){
        //处理流程
        curSum += root->val;
        res_temp.push_back(root->val);
        bool isLeaf = (root->left == nullptr && root->right == nullptr);
        if(curSum == expectNumber && isLeaf){
            res.push_back(res_temp);
        }
        //递归
        if(root->left){
            FindPathCore(res, res_temp, root->left,curSum,expectNumber);
        }
        if(root->right){
            FindPathCore(res, res_temp, root->right,curSum,expectNumber);
        }
        //状态回退
        res_temp.pop_back();
        curSum -= root->val;
    }
};
```

网页链接：二叉树中和为某一值的路径

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。

code

分解让复杂问题简单，链表，树

```
/*
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)
    {
        TreeNode* BinaryList = nullptr;
        //排序的二叉搜索树，考虑中序遍历二叉树。二维指针(指针的指针)
        ConvertCore(pRootOfTree,&BinaryList);
        //BinaryList指向双向链表的尾节点
        TreeNode* BinaryListNode = BinaryList;
        while(BinaryListNode && BinaryListNode->left){
            BinaryListNode = BinaryListNode->left;
        }
        return BinaryListNode;
    }
    void ConvertCore(TreeNode* pRootOfTree,TreeNode** BinaryList){
        //递归终止条件
        if(pRootOfTree == nullptr){
            return;
        }
        if(pRootOfTree->left){
            ConvertCore(pRootOfTree->left,BinaryList);
        }
        pRootOfTree->left = *BinaryList;
        if(*BinaryList){
            (*BinaryList)->right = pRootOfTree;
        }
        *BinaryList = pRootOfTree;
        if(pRootOfTree->right){
            ConvertCore(pRootOfTree->right,BinaryList);
        }
    }
};
```

网页链接：二叉搜索树与双向链表

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。

code

知识迁移能力,树

```
/*
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 == nullptr){
            return 0;
        }
        //递归
        //return 1+max(TreeDepth(pRoot->left),TreeDepth(pRoot->right));
        //非递归
        queue<TreeNode*> q;
        q.push(pRoot);
        int depth = 0;
        int count = 0;
        int levelCount = 1;
        while(!q.empty()){
            TreeNode* temp = q.front();
            q.pop();
            count++;
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
            if(count == levelCount){
                depth++;
                count = 0;
                levelCount = q.size();
            }
        }
        return depth;
    }
};
```

网页链接：二叉树的深度

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。

code

知识迁移能力,树

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

网页链接：平衡二叉树

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

code

树

```
/*
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) {
        string result;
        SerializeCore(root,result);
        int bufSize = result.size();
        char *res = new char[bufSize + 1];
        for(int i = 0;i < bufSize; i++){
            res[i] = result[i];
        }
        res[bufSize] = '\0';
        return res;
    }
    TreeNode* Deserialize(char *str) {
        return DeserializeCore(str);
    }
    void SerializeCore(TreeNode *root,string& str){
        if(!root){
            str += '#';
            return;
        }
        else{
            str += to_string(root->val);
            str += ',';
            SerializeCore(root->left,str);
            SerializeCore(root->right,str);
        }
    }
    TreeNode* DeserializeCore(char* &node){
        if(*node == '#'){
            node++;
            return nullptr;
        }
        int num = 0;
        while(*node != '\0' && *node != ','){
            num = num * 10 + *node - '0';
            node++;
        }
```

```
        TreeNode* root = new TreeNode(num);
        if(*node == '\0'){
            return root;
        }
        else{
            node++;
        }
        root->left = DeserializeCore(node);
        root->right = DeserializeCore(node);
        return root;
    }
};
```

网页链接：序列化二叉树

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。

code

举例让抽象具体化，树

```
class Solution {
public:
    bool VerifySquenceOfBST(vector<int> sequence) {
        int n = sequence.size();
        if(n == 0){
            return false;
        }
        if(n == 1){
            return true;
        }
        return helper(sequence,0,n-1);
    }
    bool helper(vector<int> sequence,int start,int end){
        if(start >= end){
            return true;
        }
        int root = sequence[end]; //后序遍历的最后一个节点是根节点
        int index = start;
        for(index = start; index < end; index++){
            if(sequence[index] > root){
                break;
            }
```

```
        } //寻找左右子树序列的分界点
        for(int i = index; i < end; i++){
            if(sequence[i] < root){
                return false;
            }
        } //查看右子树序列照中有没有异常情况
        bool left = true;
        if(index < end){ //递归判断左子树序列
            left = helper(sequence,start,index);
        }
        bool right = true;
        if(index < end - 1){ //递归判断右子树序列
            right = helper(sequence,index+1,end);
        }
        return left && right;
    }
};
```

网页链接：二叉搜索树的后序遍历序列

二叉搜索树的第k个节点

题目描述

code

树

```
/*
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;
        }
        KthNodeCore(pRoot,k);
        return res;
    }
    void KthNodeCore(TreeNode* pRoot,int k){
        if(pRoot->left){
```

```
            KthNodeCore(pRoot->left,k);
        }
        if(k == ++count){
            res = pRoot;
            return;
        }
        if(pRoot->right){
            KthNodeCore(pRoot->right,k);
        }
    }
    int count = 0;
    TreeNode* res = nullptr;
};
```

网页链接：二叉搜索树的第k个节点

栈

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。

code

栈和队列

```
class Solution
{
public:
    void push(int node) {
        stack1.push(node);
    }
    int pop() {
        int res = 0;
        if(!stack2.empty()){
            res = stack2.top();
            stack2.pop();
        }
        else{
            while(!stack1.empty()){
                int temp = stack1.top();
                stack2.push(temp);
                stack1.pop();
            }
```

```
            res = stack2.top();
            stack2.pop();
        }
        return res;
    }
private:
    stack<int> stack1; //负责入对列
    stack<int> stack2; //负责出队列
};
```

网页链接：用两个栈实现队列

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））。

code

举例让抽象具体化，栈

```
class Solution {
public:
    void push(int value) {
        data.push(value);
        if(extra.empty() || value < extra.top()){
            extra.push(value);
        }
        else{
            extra.push(extra.top());
        }
    }
    void pop() {
        data.pop();
        extra.pop();
    }
    int top() {
        return data.top();
    }
    int min() {
        return extra.top();
    }
private:
    stack<int> data; //数据栈
    stack<int> extra; //辅助栈，用于存放最小值
};
```

网页链接：包含min函数的栈

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）

code

举例让抽象具体化，栈

```
class Solution {
public:
    bool IsPopOrder(vector<int> pushV,vector<int> popV) {
        stack<int> extra; //利用辅助栈模拟栈的压入弹出
        if(pushV.size() != popV.size()){
            return false;
        }
        int len = popV.size();
        int i = 0;
        int j = 0;
        while(i < len){
            while(extra.empty() ||  extra.top() != popV[i]){
                if(j > len - 1){
                    return false;
                }
                extra.push(pushV[j]);
                j++;
            }
            if(extra.top() == popV[i]){
                extra.pop();
                i++;
            }
        }
        return true;
    }
};
```

网页链接：栈的压入、弹出序列

回溯

矩阵中的路径

题目描述

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。

code

回溯法

```
class Solution {
public:
    bool hasPath(char* matrix, int rows, int cols, char* str)
    {
        if(matrix == nullptr) return false;
        if(str == nullptr) return true;
        vector<bool> visited(rows*cols,false);
        bool res = false;
        for(int i = 0; i < rows && !res; i++){
            for(int j = 0; j < cols && !res; j++){
               if(matrix[i*cols+j] == str[0] && visited[i*cols+j]== false){
                   res = res || helper(matrix,str,visited,i,j,rows,cols,0);
               }
            }
        }
        return res;
    }
    bool helper(char* matrix,char* str,vector<bool>& visited,int i,int
j,int rows,int cols,int num){
        if(num == strlen(str)){
            return true;
        }
        if(i<0 || i>=rows || j<0 || j>=cols || visited[i*cols+j] == true){
            return false;
        }
        bool res = false;
        if(matrix[i*cols+j] == str[num]){
            visited[i*cols+j] = true;
            res = res || helper(matrix,str,visited,i-1,j,rows,cols,num + 1)
|| \
                helper(matrix,str,visited,i+1,j,rows,cols,num+1) || \
                helper(matrix,str,visited,i,j-1,rows,cols,num+1) || \
                helper(matrix,str,visited,i,j+1,rows,cols,num+1);
            visited[i*cols+j] = false;
```

```
        }
        return res;
    }
};
```

网页链接：矩阵中的路径

机器人的运动路径

问题描述

> 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？

code

回溯法

```
class Solution {
public:
    int movingCount(int threshold, int rows, int cols)
    {
        int count = 0;
        if(threshold <= 0 || (rows <= 0 && cols <= 0)){
            return 0;
        }
        vector<vector<bool>> visited(rows,vector<bool>(cols,false));
        helper(threshold,0,0,rows,cols,count,visited);
        return count;
    }
    void helper(int threshold, int row, int col,int rows,int cols,int&
count, vector<vector<bool>>& visited){
        if(!isValid(threshold,row,col,rows,cols,visited)){
            return;
        }
        visited[row][col] = true;
        count++;
        helper(threshold,row-1,col,rows,cols,count,visited);
        helper(threshold,row+1,col,rows,cols,count,visited);
        helper(threshold,row,col-1,rows,cols,count,visited);
        helper(threshold,row,col+1,rows,cols,count,visited);
        //visited[row][col] = false;
    }
    bool isValid(int threshold, int row, int col,int rows, int cols,
vector<vector<bool>>& visited){
        return row>=0 && row< rows && col>=0 && col < cols && visited[row]
```

```
[col] == false && digitSum(row) + digitSum(col) <= threshold;
    }
    int digitSum(int num){
        int res = 0;
        while(num){
            res += num % 10;
            num /= 10;
        }
        return res;
    }
};
```

网页链接：机器人的运动路径

动态规划

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

code

动态规划、递归和循环

状态转移方程： $f(n) = f(n-1) + f(n-2)$

```
class Solution {
public:
    int Fibonacci(int n) {
        if(n == 0){
            return 0;
        }
        if(n == 1 || n == 2){
            return 1;
        }
        int n1 = 1;
        int n2 = 1;
        for(int i = 3; i <= n; i++){
            int temp = n1;
            n1 = n2;
            n2 = temp + n2;
        }
        return n2;
```

```
    }
};
```

网页链接：斐波那契数列

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）

code

动态规划，递归和循环

状态转移方程： $f(n) = f(n-1) + f(n-2)$

```
class Solution {
public:
    int jumpFloor(int number) {
        if(number == 1 || number == 2){
            return number;
        }
        int f1 = 1;
        int f2 = 2;
        for(int i = 3; i <= number; i++){
            int temp = f1;
            f1 = f2;
            f2 = temp + f1;
        }
        return f2;
    }
};
```

网页链接：跳台阶

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。

code

动态规划，递归和循环

状态转移方程： $f(n) = f(n-1) + f(n-2) + ... + f(0)$ ， $f(n-1) = f(n-2) + f(n-3) + ... + f(0)$ ，*两式相减，有* $f(n) = 2*f(n-1)$

最终的状态转移方程： $f(n) = 2*f(n-1)$

```
class Solution {
public:
    int jumpFloorII(int number) {
        if(number == 1 || number == 2){
            return number;
        }
        int res = 2;
        for(int i = 3; i<= number;i++){
            res *= 2;
        }
        return res;
    }
};
```

网页链接：变态跳台阶

矩形覆盖

题目描述

我们可以用2*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2*1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？

code

动态规划，递归和循环

状态转移方程： $f(n) = f(n-1) + f(n-2)$

```
class Solution {
public:
    int rectCover(int number) {
        if(number <= 2){
            return number;
        }
        int f1 = 1;
        int f2 = 2;
        for(int i = 3; i<= number; i++){
```

```
            int temp = f1;
            f1 = f2;
            f2 = temp + f1;
        }
        return f2;
    }
};
```

网页链接：矩形覆盖

其他

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）。

code

时间效率

```
class Solution {
public:
    int NumberOf1Between1AndN_Solution(int n)
    {
        /*
        if( n < 1){
            return 0;
        }
        if( n <= 9 ){
            return 1;
        }
        int high = 0;
        int cnt = 0;
        int k = 0;
        int cur = 0;
        //如果第 i 位(自右向左，从1开始标号)上的数字是0，则第 i 位可能出现 1 的次数由更高位决定(若没有高位，则视高位为0)，等于更高位数乘以当前位数的权重10^(i-1)

       //如果第 i 位上的数字为 1，则第 i 位上出现 1 的次数不仅受更高位影响，还受低位影响(若没有低位，视低位为0)，等于更高位数乘以当前位数的权重 10^(i-1) + (低位数 + 1)
```

```
        //如果第 i 位上的数字大于 1，则第 i 位上可能出现 1 的次数仅由更高位决定(若没有高位，视高位为0)，等于(更高位数 + 1)乘以当前位数的权重 10^(i-1)

        for( int i = 1; k = n / i; i *= 10 ){
            high = k / 10; //在i位之前的数字（如1235，十位之间的数字为12） ，high表示i位的该位
            cnt += high * i;  //更高位数乘以当前位数的权重 10^(i-1)
            cur = k % 10;//i位的值（如1235，百位的数字为3）
            if(cur > 1)
                cnt += i; //1乘以当前位数的权重 10^(i-1)
            else if(cur == 1)
                cnt += n - k * i + 1; //n - k * i表示i位的低位
        }
        return cnt;
        */
        char str[50];
        sprintf(str,"%d",n);
        return NumberOf1(str);
    }
    int NumberOf1(const char* str){
        if(!str || *str < '0' || *str > '9' || *str == '\0'){
            return 0;
        }
        int first = *str - '0';
        unsigned int length = static_cast<unsigned int>(strlen(str));
        if( length == 1 && first == 0){
            return 0;
        }
        if( length == 1 && first <= 9){
            return 1;
        }
        int numofFirstDights = 0;
        if(first > 1){
            numofFirstDights = pow(10,length - 1); //最高位为1的数字个数
        }
        else if(first == 1){
            numofFirstDights = stoi(str + 1) + 1; //stoi(str + 1)表示去除最高位后的数字
        }
        int numofExtraDigits = 0;
        numofExtraDigits = first * (length - 1) * pow(10,length - 2); //除第1位外其余位为1的次数
        int numRecur = NumberOf1(str + 1); //去除最高位的数字
        return numofFirstDights+numofExtraDigits+numRecur;
    }
};
```

网页链接：整数中1出现的次数

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.

code

画图让抽象问题形象化

```
class Solution {
public:
        vector<int> printMatrix(vector<vector<int> > matrix) {
                vector<int> res;
                int m = matrix.size(); //行数
                if (m == 0){
                        return res;
                }
                int n = matrix[0].size(); //列数
                int i = 0;
                int j = 0;
                while (i * 2 < m && j * 2 < n){
                        printMatrixCircle(matrix, res, i, j, m, n);
                        ++i;
                        ++j;
                }
                return res;
        }
        void printMatrixCircle(vector<vector<int> > matrix, vector<int>&
result, const int i,const int j, const int m, const int n){
                int endX = n - i - 1;
                int endY = m - j - 1;
                //从左到右打印一行
                for (int k = i; k <= endX; k++){
                        result.push_back(matrix[j][k]);
                }
                //从上到下打印一列
                if (j < endY){
                        for (int l = j + 1; l <= endY; l++){
                                result.push_back(matrix[l][endX]);
                        }
                }
                //从右到左打印一行
                if (i < endX && j < endY){
                        for (int kk = endX - 1; kk >= i; kk--){
                                result.push_back(matrix[endY][kk]);
                        }
                }
                //从下到上打印一列
                if (i < endX && j < endY - 1){
```

```
                        for (int ll = endY - 1; ll > j; ll--){
                                result.push_back(matrix[ll][i]);
                        }
                }
        }
};
```

网页链接：顺时针打印矩阵

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。

code

时间空间效率的平衡

```
class Solution {
public:
    int GetUglyNumber_Solution(int index) {
        if(index < 1){
            return 0;
        }
        int* UglyNumber = new int[index];
        UglyNumber[0] = 1;
        int UglyNumberOf2 = 0;
        int UglyNumberOf3 = 0;
        int UglyNumberOf5 = 0;
        int cur = 1;
        while(cur < index){
            int next = getMin(UglyNumber[UglyNumberOf2] * 2,
UglyNumber[UglyNumberOf3] * 3,UglyNumber[UglyNumberOf5] * 5);
            UglyNumber[cur] = next;
            while(UglyNumber[UglyNumberOf2] * 2 <= next){
                ++UglyNumberOf2;
            }
            while(UglyNumber[UglyNumberOf3] * 3 <= next){
                ++UglyNumberOf3;
            }
            while(UglyNumber[UglyNumberOf5] * 5 <= next){
                ++UglyNumberOf5;
            }
            ++cur;
        }
        int res = UglyNumber[index-1];
```

```
            delete[] UglyNumber;
            UglyNumber = nullptr;
            return res;
        }
        int getMin(int a, int b, int c){
            int temp = a > b ? b : a;
            return temp > c ? c : temp;
        }
    };
```

网页链接：丑数

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。

code

位运算

```
class Solution {
public:
     int  NumberOf1(int n) {
         int num = 0;
         while(n){
             n &= (n-1);
             num++;
         }
         return num;
     }
};
```

网页链接：二进制中1的个数

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。

code

代码的完整性

```
class Solution {
public:
    double Power(double base, int exponent) {
        bool isNeg = false;
        if(exponent < 0){
            isNeg = true;
            exponent = abs(exponent);
        }
        //递归终止条件
        if(exponent == 0){
            return 1;
        }
        double res;
        if(exponent & 1){ //exponent为奇数
            res = Power(base,exponent/2) * Power(base,exponent/2) * base;
        }
        else{
            res = Power(base,exponent/2) * Power(base,exponent/2);
        }
        return isNeg? 1.0 / res : res;
    }
};
```

网页链接：数值的整数次方

圆圈中最后剩下的数(约瑟夫环)

题目描述

0, 1, … , n-1 这 n 个数字排成一个圈圈，从数字 0 开始每次从圆圈里删除第 m 个数字。求出这个圈圈里剩下的最后一个数字。

code

抽象建模能力

```
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++){
```

```
            //f(n) = [f(n-1) + m] % i
            last = (last + m) % i;
        }
        return last;
    }
};
```

网页链接：圆圈中最后剩下的数

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）

code

发散思维能力

```
class Sum{
public:
    Sum(){
        ++n;
        sum += n;
    }
    static int GetSum(){
        return sum;
    }
    static void Reset(){
        n = 0;
        sum = 0;
    }
private:
    static int n;
    static int sum;
};
int Sum::n = 0;
int Sum::sum = 0;
class Solution {
public:
    int Sum_Solution(int n) {
        Sum::Reset();
        Sum* a = new Sum[n];
        delete[] a;
        a = nullptr;
        return Sum::GetSum();
```

```
    }
};
```

网页链接：求1+2+3+...+n

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。

code

发散思维能力

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

网页链接：不用加减乘除做加法