1. 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
- 关键：数字都在0到n-1的范围内

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

2. 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。
- 思路：注意数组是有顺序的。从右上角开始比较，大于target就只可能在左边，小于就只能在下面。若从中间开始找就会很麻烦。

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

```
        int m = array.size(), n = array[0].size();
        int r = 0, c = n-1;
        while(r < m && c >= 0){
            if(array[r][c] == target)
                return true;
            if(array[r][c] > target)
                c--;
            else
                r++;
        }
        return false;
    }
};
```

3. 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。
- O(n)复杂度，先遍历一遍根据空格数计算出替换后的字符串长度，再从后往前复制字符。

```
class Solution {
public:
    void replaceSpace(char *str,int length) {
        int i = 0, j = 0, sum = 0;
        while(str[i] != 0){
            if(str[i++] == ' ')
                sum++;
        }
        j = i + sum * 2; //空格换为%20后的长度
        while(i >= 0){//从后往前替换
            if(str[i] != ' ')
                str[j--] = str[i--];
            else{
                str[j--] = '0';
                str[j--] = '2';
                str[j--] = '%';
                i--;//跳过空格
            }
        }
    }
};
```

4. 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。
- 思路：用栈，先进后出

```
```

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

5. 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。
- 递归

```
/**
 * 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) {
        return build_tree(pre, 0, pre.size()-1, vin, 0, vin.size()-1);
    }
    TreeNode* build_tree(vector<int> &pre, int pre_l, int pre_r, vector<int> &vin, int vin_l, int vin_r){
        if(pre_l > pre_r)
            return nullptr;
        if(pre_l == pre_r)
            return new TreeNode(pre[pre_l]);
        TreeNode *root = new TreeNode(pre[pre_l]);
```

```
        int i = 0;
        for(;i+vin_l<=vin_r; i++){
            if(vin[i+vin_l] == pre[pre_l])
                break;
        }
        root->left = build_tree(pre, pre_l+1, pre_l+i, vin, vin_l, vi
        root->right = build_tree(pre, pre_l+i+1, pre_r, vin, vin_l+i+
        return root;
    }
};
```

6. 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。
- 解法：分两种情况，有右节点和没有右节点。有右节点：下一个节点就是右子树的最左边的节点。没有右节点：下一个节点就是（是父亲节点左子树的节点）

```
/*
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->right){
            TreeLinkNode* tmp = pNode->right;
            while(tmp->left)
                tmp = tmp->left;
            return tmp;
        }
        while(pNode->next && pNode->next->left != pNode){// 防止pNode-
            pNode = pNode->next;
        }
        return pNode->next;
    }
};
```

7. 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。

- 思路：一个栈是先进后出，从一个栈出栈再入到第二个栈，第二个栈就是先进先出了，就完成了模拟队列。

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

    int pop() {
        if(stack2.empty()){
        //stack2为空，就把stack1中的元素入栈到stack2中
            while(!stack1.empty()){
                stack2.push(stack1.top());
                stack1.pop();
            }
        }
        int ans = stack2.top();
        stack2.pop();
        return ans;
    }

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

8. 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
- O(n)时间复杂度

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

    }
};
```

9. 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）。
- 解法：其实就是斐波那契数列，第n级台阶可以由第n-1级跳一步达到，也可以有第n-2级跳两步达到。

```
class Solution {
public:
    int jumpFloor(int number) {
        if(number == 1) return 1;
        int a = 1, b = 1, i = 1, tmp = 0;
        while(i++ < number){
            tmp = a; a = b; b = tmp + b;
        }
        return b;
    }
};
```

10. 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。

```
class Solution {
public:
    int jumpFloorII(int number) {
        return 1 << (number - 1);
    }
};
```

11. 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?
- 解法：也是斐波那契数列

12. 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。
- 二分法查找

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

13. 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。
- 回溯法 -- 隐式图的深度优先搜索

```
class Solution {
public:
    bool hasPath(char* matrix, int rows, int cols, char* str)
    {
        bool *visit = new bool[rows * cols];
        memset(visit, 0, sizeof(bool) * rows * cols);
        for(int r=0; r<rows; r++){
            for(int c=0; c<cols; c++){
                if(judge(matrix, rows, cols, r, c, str, 0, visit)){
                    delete[] visit;
                    return true;
                }
            }
        }
        delete[] visit;
        return false;
    }
    bool judge(char *matrix, int rows, int cols, int r, int c,
               char *str, int idx, bool *visit){
        if(str[idx] == '\0') return true;
        if(r < 0 || r >= rows || c < 0 || c >= cols) return false;
        if(visit[r*cols+c]) return false;
        if(matrix[r*cols+c] != str[idx]) return false;
        visit[r*cols+c] = true;
```

```
        bool has_subpath = judge(matrix, rows, cols, r+1, c, str, idx
            || judge(matrix, rows, cols, r-1, c, str, idx+1, visit)
            || judge(matrix, rows, cols, r, c+1, str, idx+1, visit)
            || judge(matrix, rows, cols, r, c-1, str, idx+1, visit);
        visit[r*cols+c] = false;
        return has_subpath;
    }
};
```

14. 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?
- 回溯法

```
class Solution {
public:
    int movingCount(int threshold, int rows, int cols)
    {
        bool *visit = new bool[rows * cols];
        memset(visit, 0, sizeof(bool)*rows*cols);
        int count = moving_count(threshold, rows, cols, 0, 0, visit);
        return count;
    }
    int moving_count(int threshold, int rows, int cols, int r, int c,
        int count = 0;
        if(r<0 || r>=rows || c<0 || c>=cols || visit[r*cols+c]) retur
        if(check(threshold, r, c)){
            visit[r*cols+c] = true;
            count = 1 + moving_count(threshold, rows, cols, r+1, c, v
                + moving_count(threshold, rows, cols, r-1, c, visit)
                + moving_count(threshold, rows, cols, r, c+1, visit)
                + moving_count(threshold, rows, cols, r, c-1, visit);

        }
        return count;
    }
    bool check(int threshold, int r, int c){
        int sum = 0;
        while(r){
            sum += r % 10;
            r /= 10;
        }
        while(c){
            sum += c % 10;
            c /= 10;
        }
        return sum <= threshold;
    }
};
```

15. 剪绳子（待定）

16. 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。
- 解法：从低位依次右移数一，不要从高位往低位数，因为负数的符号位右移一直是1。

```
// 解法一
class Solution {
public:
     int  NumberOf1(int n) {
         int a = 1, sum = 0;
         while(a){
             if(n & a)
                 sum++;
             a <<= 1;
         }
         return sum;
     }
};
// 解法二  例：1100-1=1011,  1100 & 1011 != 0
class Solution {
public:
     int  NumberOf1(int n) {
         int a = 1, sum = 0;
         while(n){
             sum++;
             n = (n-1) & n;;
         }
         return sum;
     }
};
```

17. 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。
- O(logn).要考虑负数的情况

```
class Solution {
public:
    double Power(double base, int exponent) {
        if(exponent == 1)
            return base;
        if(exponent == 0)
            return 1;
        if(base == 0)
            return 0;//防止出现1/0
        int flag = 0;
```

```
        if(exponent < 0){
            flag = 1;
            exponent *= -1;
        }
        double ans = Power(base, exponent>>1);
        ans *= ans;
        if(exponent & 1)
            ans *= base;
        if(flag)
            ans = 1 / ans;
        return ans;
    }
};
```

18. 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**。
- 冒泡排序保持稳定性，插入排序也可以。注意，有没有黑体字的解法是不一样的。

```
class Solution {
public:
    void reOrderArray(vector<int> &array) {
        int size = array.size();
        for(int i=0; i<size; i++){
            int isSwap = 0;
            for(int j=size-1; j>i; j--){
                if((array[j]&1) == 1 && (array[j-1]&1) == 0){
                    swap(array[j], array[j-1]);
                    isSwap = 1;
                }
            }
            if(!isSwap)
                break;
        }
    }
};
```

19. 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。
- 双指针，让一个指针先走k-1步，再两个指针一起走。简单题要考虑鲁棒性。

```
/*
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(k == 0 || pListHead == nullptr) return nullptr;
        ListNode *left = pListHead, *right = pListHead;
        while(right != nullptr && --k){
            right = right->next;
        }
        if(k) return nullptr;
        while(right->next){
            left = left->next;
            right = right->next;
        }
        return left;
    }
};
```

20. 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。

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
        val(x), next(NULL) {
    }
};
*/
class Solution {
public:
    ListNode* EntryNodeOfLoop(ListNode* pHead)
    {
        ListNode *fast = pHead, *slow = pHead;
        int flag = 0;
        while(fast && fast->next){
            fast = fast->next->next;
            slow = slow->next;
            if(fast == slow){
                flag = 1;
                break;
            }
        }
        if(flag == 0) return nullptr;
        slow = pHead;
        while(fast != slow){
            fast = fast->next;
            slow = slow->next;
```

```
        }
        return fast;
    }
};
```

21. 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。

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

22. 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。

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* Merge(ListNode* pHead1, ListNode* pHead2)
    {
```

```
        ListNode *head = new ListNode(0);
        ListNode *tmp = head;
        while(pHead1 && pHead2){
            if(pHead1->val < pHead2->val){
                tmp->next = pHead1;
                pHead1 = pHead1->next;
            }
            else{
                tmp->next = pHead2;
                pHead2 = pHead2->next;
            }
            tmp = tmp->next;
        }
        if(pHead1) tmp->next = pHead1;
        else tmp->next = pHead2;
        return head->next;
    }
};
```

23. 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）

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

```
    }
};
```

24. 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。
- 扩展：用循环来做（模拟栈）。

```
class Solution {
public:
    void Mirror(TreeNode *pRoot) {
        if(pRoot){
            TreeNode *tmp = pRoot->left;
            pRoot->left = pRoot->right;
            pRoot->right = tmp;
            Mirror(pRoot->right);
            Mirror(pRoot->left);
        }
    }
};
```

由于递归的本质是编译器生成了一个函数调用的栈，因此用循环来完成同样任务时，最简单的办法就是用一个辅助栈来模拟递归。首先把树的头结点放入栈中。在循环中，只要栈不为空，弹出栈的栈顶结点，交换它的左右子树。如果它有左子树，把它的左子树压入栈中；如果它有右子树，把它的右子树压入栈中。这样在下次循环中就能交换它儿子结点的左右子树了。

```
void MirrorIteratively(BinaryTreeNode* pRoot)
{
    if (pRoot == NULL)
        return;

    std::stack<BinaryTreeNode*> stackTreeNode;
    stackTreeNode.push(pRoot);

    while (stackTreeNode.size() > 0)
    {
        BinaryTreeNode *pNode = stackTreeNode.top();
        stackTreeNode.pop();

        BinaryTreeNode *pTemp = pNode->m_pLeft;
        pNode->m_pLeft = pNode->m_pRight;
        pNode->m_pRight = pTemp;

        if (pNode->m_pLeft)
            stackTreeNode.push(pNode->m_pLeft);

        if (pNode->m_pRight)
            stackTreeNode.push(pNode->m_pRight);
    }
}
```

25. 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。

```
class Solution {
public:
    bool isSymmetrical(TreeNode* pRoot)
    {
        if(!pRoot) return true;
        return judge(pRoot->left, pRoot->right);
    }
    bool judge(TreeNode* left, TreeNode* right){
        if(!left && !right) return true;
        if((!left) ^ (!right)) return false;
        if(left->val != right->val) return false;
        return judge(left->left, right->right)
            && judge(left->right, right->left);
    }
};
```

26. 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.

```
class Solution {
public:
    vector<int> printMatrix(vector<vector<int> > matrix) {
        vector<int> ans;
        if(matrix.empty()) return ans;
        int rows = matrix.size(), cols = matrix[0].size();
        int r = 0, c = 0, left = 0, up = 0, right = cols-1, down = rov
        while(1){
            for(c = up; c<=right; c++)
                ans.push_back(matrix[up][c]);
            if(++up > down) break;
            for(r = up; r<=down; r++)
                ans.push_back(matrix[r][right]);
            if(--right < left) break;
            for(c = right; c>=left; c--){
                ans.push_back(matrix[down][c]);
            }
            if(--down < up) break;
            for(r = down; r>=up; r--){
                ans.push_back(matrix[r][left]);
            }
```

```
            if(++left > right) break;
        }
        return ans;
    }
};
```

27. 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））。

```
class Solution {
public:

    stack<int> st;
    stack<int> minst;

    void push(int value) {
        st.push(value);
        if(minst.empty() || minst.top()>value) minst.push(value);
        else minst.push(minst.top());
    }
    void pop() {
        if(!st.empty()){
            st.pop();
            minst.pop();
        }
    }
    int top() {
        return st.top();
    }
    int min() {
        return minst.top();
    }
};
```

28. 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）

```
class Solution {
public:
    bool IsPopOrder(vector<int> pushV,vector<int> popV) {
        stack<int> st;
        int i = 0, j = 0, size1 = pushV.size(), size2 = popV.size();
```

```
        if(size1 != size2) return false;
        for(; i<size1; i++){
            st.push(pushV[i]);
            while(st.top() == popV[j]){
                st.pop();
                j++;
                if(st.empty()) break;
            }
        }
        return st.empty();
    }
};
```

29. 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。

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

30. 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。

- 解法：后序遍历根节点在最后，根据根节点将数组分为左右两部分（若无法分成两部分，则不是搜索树），再递归检查左右子树。

```
class Solution {
public:
    bool VerifySquenceOfBST(vector<int> sequence) {
        if(sequence.empty()) return false;
        return judge(sequence, 0, sequence.size()-1);
    }
    bool judge(vector<int> &vec, int left, int right){
        if(left >= right) return true;
        int root = vec[right];
        int i=0, j=right-1;
        while(vec[i] < root) i++;
        while(vec[j] > root) j--;
        if(i <= j) return false;
        return judge(vec, left, i-1) && judge(vec, j+1, right-1);
    }
};
```

31. 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。

```
class Solution {
public:
    vector<vector<int> > ans;
    vector<int> tmp;
    vector<vector<int> > FindPath(TreeNode* root,int expectNumber) {
        if(root == nullptr) return ans;
        dfs(root, expectNumber);
        return ans;
    }
    void dfs(TreeNode* root,int expectNumber){
        tmp.push_back(root->val);
        if(root->val == expectNumber && !root->left && !root->right)
            ans.push_back(tmp);
        expectNumber -= root->val;
        if(root->left) dfs(root->left, expectNumber);
        if(root->right) dfs(root->right, expectNumber);
        tmp.pop_back();
    }
};
```

32. 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）

- 方法一：先按next复制节点，用哈希表将复制前后的节点一一对应起来，这样就可以用O(1)的时间复杂度找到random节点。
- 方法二：第一步，将复制节点连在旧节点后面；第二步，复制random节点；第三步，剪开。

```
/*
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) return nullptr;
        CloneNode(pHead);
        ConnectRandomNode(pHead);
        return ReconnectNode(pHead);
    }
    void CloneNode(RandomListNode* pHead){
        while(pHead){
            RandomListNode* pClone = new RandomListNode(pHead->label)
            pClone->next = pHead->next;
            pHead->next = pClone;
            pHead = pClone->next;
        }
    }
    void ConnectRandomNode(RandomListNode* pHead){
        while(pHead){
            if(pHead->random)
                pHead->next->random = pHead->random->next;
            pHead = pHead->next->next;
        }
    }
    RandomListNode* ReconnectNode(RandomListNode* pHead){
        RandomListNode* pCloneHead = pHead->next;
        RandomListNode* pCloneNode = pCloneHead;
        pHead->next = pCloneHead->next;
        pHead = pHead->next;
        while(pHead){
            pCloneNode->next = pHead->next;
            pHead->next = pHead->next->next;
            pHead = pHead->next;
            pCloneNode = pCloneNode->next;
        }
        return pCloneHead;
    }
};
```

33. 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。
- 众所周知，BST的中序遍历是顺序的。我想到的方法是：中序遍历，将树的节点依次存入一个数组，再把节点连起来。书上给的方法不需要额外的空间。

```
/*
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) return nullptr;
        TreeNode *pLastNodeInList = nullptr;
        ConvertNode(pRootOfTree, &pLastNodeInList);

        while(pLastNodeInList && pLastNodeInList->left){
            pLastNodeInList = pLastNodeInList->left;
        }
        return pLastNodeInList;
    }
    void ConvertNode(TreeNode* root, TreeNode** pLastNodeInList){
        if(root->left)//先找到最左端的叶子结点，连接左子树
            ConvertNode(root->left, pLastNodeInList);
        root->left = *pLastNodeInList;
        if(*pLastNodeInList)
            (*pLastNodeInList)->right = root;
        *pLastNodeInList = root;
        if(root->right){//连接右子树
            ConvertNode(root->right, pLastNodeInList);
        }
    }
};
```

34. 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。

```
class Solution {
private:
    vector<int> max;
    vector<int> min;
public:
    void Insert(int num)
    {//为保持平衡，偶数放右侧，奇数放左侧
        if(((min.size()+max.size()) & 1) == 0){//偶数放入右侧，最小堆
            if(max.size() && num < max[0]){//num小于左侧最大值，要放入左侧
                max.push_back(num);
                push_heap(max.begin(), max.end(), less<int>());
                num = max[0];
                pop_heap(max.begin(), max.end(), less<int>());
                max.pop_back();
            }
            min.push_back(num);
            push_heap(min.begin(), min.end(), greater<int>());
        }
        else{//奇数放入左侧，最大推
            if(max.size() && num > max[0]){// num大于右侧最小值，要放入右
                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 sum = max.size() + min.size();
        double mid = 0;
        if(sum & 1) mid = min[0];
        else mid = (min[0] + max[0]) / 2.0;
        return mid;
    }
};
```

35. 连续子数组的最大和

- 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)。

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

```
        }
        return max;
    }
};
```

36. 整数中1出现的次数

- 参考

```
class Solution {
public:
    int NumberOf1Between1AndN_Solution(int n)
    {
        int count = 0;
        for (int i = 1; i <= n; i *= 10) {
            int a = n / i, b = n % i;
            //之所以补8，是因为当百位为0，则a/10==(a+8)/10，
            //当百位>=2，补8会产生进位位，效果等同于(a/10+1)
            count += (a + 8) / 10 * i + ((a % 10 == 1) ? b + 1 : 0);
        }
        return count;
    }
};
```

37. 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。
- 建立数组，空间换时间

```
class Solution {
public:
    int GetUglyNumber_Solution(int index) {
        if(index <= 0) return 0;
        int *pUglyNumber = new int[index];
        pUglyNumber[0] = 1;
        int *pMulti2 = pUglyNumber;
        int *pMulti3 = pUglyNumber;
        int *pMulti5 = pUglyNumber;
        int idx = 1;
        while(idx < index){
            int min = Min(*pMulti2 * 2, *pMulti3 * 3, *pMulti5 * 5);
            pUglyNumber[idx] = min;

            if(*pMulti2 * 2 <= min) pMulti2++;
            if(*pMulti3 * 3 <= min) pMulti3++;
            if(*pMulti5 * 5 <= min) pMulti5++;
            idx++;
        }
        return pUglyNumber[index-1];
```

```
    }
    int Min(int num1, int num2, int num3){
        int min = num1 < num2 ? num1 : num2;
        min = min < num3 ? min : num3;
        return min;
    }
};
```

38. 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
- 归并排序，时间复杂度O(logN).

```
class Solution {
public:
    int InversePairs(vector<int> data) {
        if(data.empty()) return 0;

        int len = data.size();
        vector<int> copy(data);

        int count = InversePairs(data, copy, 0, len-1);
        return count;
    }
    int InversePairs(vector<int> &data, vector<int> &copy, int start,
        if(start == end){
            return 0;
        }
        int len = (end - start) >> 1;
        int left = InversePairs(copy, data, start, start+len);
        int right = InversePairs(copy, data, start+len+1, end);

        int i = start+len, j = end, idx_copy = end, count = 0;
        while(i >= start && j >= start+len+1){
            if(data[i] > data[j]){
                count = (count + j - (start + len)) % 1000000007;
                copy[idx_copy--] = data[i--];
            }
            else{
                copy[idx_copy--] = data[j--];
            }
        }
        while(i >= start) copy[idx_copy--] = data[i--];
        while(j >= start+len+1) copy[idx_copy--] = data[j--];
        return (count + left + right) % 1000000007;
    }
};
```

39.两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。
- 解法：长的链表先走几步，然后找第一个公共节点

```
/*
struct ListNode {
    int val;
    struct ListNode *next;
    ListNode(int x) :
            val(x), next(NULL) {
    }
};*/
class Solution {
public:
    ListNode* FindFirstCommonNode( ListNode* pHead1, ListNode* pHead2
        int len1 = getLen(pHead1);
        int len2 = getLen(pHead2);
        int DoL = len1 - len2;
        ListNode *pLongHead = pHead1, *pShortHead = pHead2;
        if(len1 < len2){
            pLongHead = pHead2;
            pShortHead = pHead1;
            DoL = -DoL;
        }
        for(int i=0; i<DoL; i++)
            pLongHead = pLongHead->next;
        while(pLongHead != pShortHead){
            pLongHead = pLongHead->next;
            pShortHead = pShortHead->next;
        }
        return pLongHead;
    }
    int getLen(ListNode* pHead){
        int len = 0;
        while(pHead){
            pHead = pHead->next;
            len++;
        }
        return len;
    }
};
```

40. 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。
- 二分法

```
class Solution {
public:
    int GetNumberOfK(vector<int> data ,int k) {
        if(data.empty()) return 0;
```

```
        int len = data.size();
        int left = findLeft(data, k, 0, len-1);
        int right = findRight(data, k, 0, len-1);
        if(left == -1) return 0;
        return right - left + 1;
    }
    int findLeft(vector<int> data ,int k, int start, int end){
        if(start > end) return -1;
        int mid = (start + end) >> 1;
        if(data[mid] > k)
            end = mid - 1;
        else if(data[mid] < k)
            start = mid + 1;
        else{
            if(mid == 0 || data[mid-1] < k)
                return mid;
            else
                end = mid - 1;
        }
        return findLeft(data, k, start, end);
    }
    int findRight(vector<int> data ,int k, int start, int end){
        if(start > end) return -1;
        int mid = (start + end) >> 1;
        if(data[mid] > k)
            end = mid - 1;
        else if(data[mid] < k)
            start = mid + 1;
        else{
            if(mid == data.size()-1 || data[mid+1] > k)
                return mid;
            else
                start = mid + 1;
        }
        return findRight(data, k, start, end);
    }
};
```

41. 二叉树的深度

- 这个也算后序遍历。。。树的深度等于左右子树深度加一。

```
/*
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;
        int left = TreeDepth(pRoot->left);
        int right = TreeDepth(pRoot->right);
        return 1 + (left > right ? left : right);
    }
};
```

42. 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。
- 解法：用后序遍历的方式，先求出左右子树的高度，判断左右子树均为平衡二叉树后，再判断左右子树的高度差是否小于2

```
class Solution {
public:
    bool IsBalanced_Solution(TreeNode* pRoot) {
        if(IsBalanced(pRoot) == -1) return false;
        return true;

    }
    int IsBalanced(TreeNode* pRoot){
        if(pRoot == nullptr) return 0;
        int left = IsBalanced(pRoot->left);
        int right = IsBalanced(pRoot->right);
        if(left == -1 || right == -1) return -1;
        if(abs(left - right) <= 1) return 1 + (left > right ? left :
        else return -1;
    }
};
```

43. 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。
- 解法：因为有两个不同的数字，所有数字异或完后必不为0；再根据异或完的数某一位的1，将数组中的数分成两组，再分别异或。

```
class Solution {
public:
    void FindNumsAppearOnce(vector<int> data,int* num1,int *num2) {
        int len = data.size();
        if(len < 2) return;

        int result = 0;
        for(auto x : data)
            result ^= x;
        result = find_first_one(result);
```

```
        *num1 = *num2 = 0;
        for(auto x : data){
            if(x & result) *num1 ^= x;
            else *num2 ^= x;
        }
    }
    int find_first_one(int a){
        int b = 1;
        while((a & b) == 0){
            b <<= 1;
        }
        return b;
    }
};
```

44. 和为S的连续正数序列

- 题目描述：
  输入一个正数s,打印出所有和为s的连续正数序列（至少含有两个数）。例如输入15，由于1+2+3+4+5=4+5+6=7+8=15；所以打印出三个连续序列1~5 , 4~6, 7~8；

```
class Solution {
public:
    vector<vector<int> > FindContinuousSequence(int sum) {
        vector<vector<int> > ans;
        if(sum < 3) return ans;
        int left = 1, right = 2, s = 3, mid = (sum+1)>>1;
        while(left < mid){
            if(s < sum){
                s += ++right;
            }
            else if(s > sum){
                s -= left++;
            }
            else{
                vector<int> tmp;
                for(int i=left; i<=right; i++)
                    tmp.push_back(i);
                ans.push_back(tmp);
                s += ++right;
                s -= left++;
                s -= left++;
            }
        }
        return ans;
    }
};
```

45. 翻转单词顺序列

- 题目描述：
  将“I am a student.”变为“student. a am I”
- 解法：先翻转整个句子，再翻转每个单词

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

46. 左旋转字符串

- 例如，字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。

```
class Solution {
public:
    string LeftRotateString(string str, int n) {
        int len = str.length();
        if(n < 0 || str.empty()) return str;
        n %= len;
        Reverse(str, 0, n-1);
        Reverse(str, n, len-1);
        Reverse(str, 0, len-1);
        return str;
    }
    void Reverse(string &str, int s, int e){
        while(s < e){
            char c = str[s];
            str[s] = str[e];
            str[e] = c;
            s++, e--;
        }
    }
};
```

47. 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]}。
- 解法：deque双向队列

```
class Solution {
public:
    vector<int> maxInWindows(const vector<int>& num, unsigned int size
    {
        vector<int> ans;
        int len = num.size();
        if(size <= 0 || len < size) return ans;
        deque<int> index;
        for(int i=0; i<size; i++){
            while(!index.empty() && num[i] > num[index.back()])
                index.pop_back();
            index.push_back(i);
        }
        for(int i=size; i<len; i++){
            ans.push_back(num[index.front()]);
            while(!index.empty() && num[i] > num[index.back()])
                index.pop_back();
            if(!index.empty() && i - index.front() >= size)
                index.pop_front();
            index.push_back(i);
        }
        ans.push_back(num[index.front()]);
        return ans;
    }
};
```

48. 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。
- 思路：BST的中序遍历是顺序的。

```
/*
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 || k<=0) return nullptr;
        return findKthNode(pRoot, k);
    }
    TreeNode* findKthNode(TreeNode* pRoot, int& k){
        TreeNode* target = nullptr;
        if(pRoot->left)  //遍历左子树
            target = findKthNode(pRoot->left, k);
        if(target == nullptr){  //处理根节点
            if(k-- == 1)
                target = pRoot;
        }
        if(target == nullptr && pRoot->right)  //遍历右子树
            target = findKthNode(pRoot->right, k);
        return target;
    }
};
```

49. 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。
- 思路：不用加减乘除意思就是用位运算。用异或代替无进位的求和，用与运算代替进位。

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

50. 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。
- 思路：先累乘左边，再累乘右边。

```
class Solution {
public:
    vector<int> multiply(const vector<int>& A) {
        vector<int> ans;
        int size = A.size();
        if(size == 0) return ans;
        ans.push_back(1);
        for(int i=1; i<size; i++){
            ans.push_back(ans[i-1] * A[i-1]);
        }
        int tmp = 1;
        for(int i=size-2; i>=0; i--){
            tmp *=  A[i+1];
            ans[i] *= tmp;
        }
        return ans;
    }
};
```

51. 二叉树两结点的最低公共祖先结点

- 题目描述；
  给定二叉树中的两个结点，输出这两个结点的最低公共祖先结点（LCA）。注意，该二叉树不一定是二叉搜索树。
- 后序遍历

```
struct node {
    int data;
    struct node* left;
    struct node* right;
};
Node *LCA(Node *root, Node *p, Node *q) {
    if (!root) return NULL;
    if (root == p || root == q) return root;
    Node *L = LCA(root->left, p, q);
    Node *R = LCA(root->right, p, q);
    if (L && R) return root;  // 如果p和q位于不同的子树
    return L ? L : R;  //p和q在相同的子树，或者p和q不在子树中
}
```