示例 1：

输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3 输出：Reference of the node with value = 8 输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。

示例 2：

输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1 输出：Reference of the node with value = 2 输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3：

输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2 输出：null 输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。 解释：这两个链表不相交，因此返回 null。面试题01.01 判定字符是否唯一（简单）

---

1. 题目描述

   实现一个算法，确定一个字符串 s 的所有字符是否全都不同。

   ```
   示例 1:
   输入: s = "leetcode"
   输出: false

   示例 2:
   输入: s = "abc"
   输出: true
   ```

限制：

- 0 <= len(s) <= 100
- 如果你不使用额外的数据结构，会很加分

2. 简单实现

   ```
   class Solution {
   public:
       bool isUnique(string astr) {
           int cnt[256] = {0};
           for(int i = 0; i < astr.size(); i++)
               if(cnt[astr[i]] > 0)
                   return false;
               else
                   cnt[astr[i]]++;
           return true;
       }
   };
   ```

可改进点：

- 如果是ASCII字符，则只有128个，直接用`bool cnt[128] = {faklse};`即可
- 如果是ASCII字符，**长度大于128时，一定有重复，无需计数**
- 实际上测试用例只有26个小写字符

面试题01.02 判定是否互为字符重排（简单）

1. 题目描述

   给定两个字符串 s1 和 s2，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。

   ```
   示例 1:
   输入: s1 = "abc", s2 = "bca"
   输出: true

   示例 2:
   输入: s1 = "abc", s2 = "bad"
   输出: false
   ```

   说明：

   - 0 <= len(s1) <= 100
   - 0 <= len(s2) <= 100

2. 简单实现

   ```
   class Solution {
   public:
       bool CheckPermutation(string s1, string s2) {
           if(s1.size() != s2.size())
               return false;
           vector<int> cnt(128, 0);
           for(int i = 0; i < s1.size(); i++)
               cnt[s1[i]]++;
           for(int i = 0; i < s2.size(); i++){
               cnt[s2[i]]--;
               if(cnt[s2[i]] < 0)
                   return false;
           }
           return true;
           /*方法二：排序
           sort(s1.begin(), s1.end());
           sort(s2.begin(), s2.end());
           return s1 == s2;
           */
       }
   };
   ```

   改进点：实际上又是只有26个字母的，直接用位运算模式串记录s1和s2的字母情况，只遍历一次即可

```
class Solution {
public:
    bool CheckPermutation(string s1, string s2) {
        if(s1.size() != s2.size())
            return false;
        int n1 = 0, n2 = 0;
        for(int i = 0; i < s1.size(); i++){
            n1 += 1 << (s1[i] - 'a');
            n2 += 1 << (s2[i] - 'a');
        }
        return n1 == n2;
    }
}
```

面试题01.03 URL化（简单）

1. 题目描述

   URL化。编写一种方法，将字符串中的空格全部替换为%20。假定该字符串尾部有足够的空间存放新增字符，并且知道字符串的“真实”长度。（注：用Java实现的话，请使用字符数组实现，以便直接在数组上操作。）

```
示例1:
 输入："Mr John Smith    ", 13
 输出："Mr%20John%20Smith"

示例2:
 输入："               ", 5
 输出："%20%20%20%20%20"
```

   提示：字符串长度在[0, 500000]范围内。

2. 简单实现

```
//非原地算法
class Solution {
public:
    string replaceSpaces(string S, int length) {
        string ans = "";
        for(int i = 0; i < length; i++){//只处理到length即可
            if(S[i] != ' ') ans += S[i];
            else{
                if(length) ans += "%20";
                else break;
            }
        }
        return ans;
    }
};
//原地算法
class Solution {
public:
```

```
    string replaceSpaces(string S, int length) {
        int idx = S.size() - 1;
        for(int i = length-1; i >= 0; i--){
            if(S[i] != ' ')
                S[idx--] = S[i];
            else{
                S[idx--] = '0';
                S[idx--] = '2';
                S[idx--] = '%';
            }
        }
        if(idx >= 0)//预留的空间超出所需
            S = S.substr(idx+1, S.size()-idx);
        return S;
    }
};
//也可以先遍历一遍计算所需空间，然后通过设置'\0'修改S
```

面试题01.04 回文排列（简单）

1. 题目描述

   给定一个字符串，编写一个函数判定其是否为某个回文串的排列之一。回文串是指正反两个方向都一样的单词或短语。排列是指字母的重新排列。回文串不一定是字典当中的单词。

   ```
   示例1:

   输入: "tactcoa"
   输出: true（排列有"tacocat"、"atcocta"，等等）
   ```

2. 简单实现

```
class Solution {
public:
    bool canPermutePalindrome(string s) {
        vector<bool> cnt(128, false);//两两即可成对，用bool记录即可，存在未配对的就是true
        for(int i = 0; i < s.size(); i++)
            cnt[s[i]] = !cnt[s[i]];
        bool f = false;
        for(int i = 0; i < 26; i++){
            if(cnt[i]){
                if(f)
                    return false;
                else
                    f = true;
            }
        }
        return true;
    }
};
```

PS：此处也可以使用bitset进行记录，更方便

面试题01.05 一次编辑（中等）

1. 题目描述

   字符串有三种编辑操作:插入一个字符、删除一个字符或者替换一个字符。 给定两个字符串，编写一个函数判定它们是否只需要一次(或者零次)编辑即可相同。

```
示例 1:
输入:
first = "pale"
second = "ple"
输出: True

示例 2:
输入:
first = "pales"
second = "pal"
输出: False
```

2. 简单实现

   分类讨论即可，实际上插入和删除一个字符是等效的，如示例一，可以删first的a，还可以插second的a

```
class Solution {
public:
    bool oneEditAway(string first, string second) {
        int len1 = first.size();
        int len2 = second.size();
        if(abs(len1-len2) > 1)//长度相差超过两位，不可能一次编辑
            return false;
        else if(len1 == len2){//长度相等，可能需要替换
            bool f = false;
            while(--len1 >= 0){
                if(first[len1] != second[len1]){
                    if(f)//替换过了
                        return false;
                    f = true;
                }
            }
        }
        else{
            --len1;
            --len2;
            while(len1>=0 && len2>=0){
                if(first[len1] != second[len2]){
                    if(len1 > len2)
                        len1--;//这里执行删除
                    else if(len1 < len2)
                        len2--;//这里执行删除
                    else//删除过了
```

```
                        return false;
                }
                else{
                    --len1;
                    --len2;
                }
            }
        }
        return true;
    }
};
```

面试题01.06 字符串压缩（简单）

1. 题目描述

   字符串压缩。利用字符重复出现的次数，编写一种方法，实现基本的字符串压缩功能。比如，字符串aabcccccaaa会变为a2b1c5a3。若“压缩”后的字符串没有变短，则返回原先的字符串。你可以假设字符串中只包含大小写英文字母（a至z）。

```
示例1:
 输入："aabcccccaaa"
 输出："a2b1c5a3"

示例2:
 输入："abbccd"
 输出："abbccd"
 解释："abbccd"压缩后为"a1b2c2d1"，比原字符串长度更长。
```

   提示：字符串长度在[0, 50000]范围内。

2. 简单实现

```
class Solution {
public:
    string compressString(string S) {
        if(S.size() <= 2)//长度小于等于2的一定不需要压缩
            return S;
        string ans = "";
        int cnt = 1;
        char c = S[0];
        for(int i = 1; i < S.size(); i++){
            if(S[i] != c){
                ans += c + to_string(cnt);
                c = S[i];
                cnt = 1;
            }
            else
                cnt++;
        }
        ans += c + to_string(cnt);
        if(S.size() <= ans.size())
```

```
            return s;
        else
            return ans;
    }
};
```

面试题01.07 旋转矩阵（中等）

1. 题目描述

   给定一幅由N × N矩阵表示的图像，其中每个像素的大小为4字节，编写一种方法，将图像旋转90度。

   不占用额外内存空间能否做到?

```
示例 1:
给定 matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],
原地旋转输入矩阵，使其变为:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]

示例 2:
给定 matrix =
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],
原地旋转输入矩阵，使其变为:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

2. 简单实现

   顺时针旋转90度 = 先转置，再水平翻转

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        int n = matrix.size();
```

```
        if(n == 1)
            return;
        //转置
        for(int i = 0; i < n-1; i++)
            for(int j = i+1; j < n; j++)
                swap(matrix[i][j], matrix[j][i]);
        //水平翻转
        for(int j = 0; j < n/2; j++)
            for(int i = 0; i < n; i++)
                swap(matrix[i][j], matrix[i][n-1-j]);
    }
};
```

PS：交换可以用^实现，不占用额外存储空间

面试题01.08 零矩阵（中等）

1. 题目描述

   编写一种算法，若M × N矩阵中某个元素为0，则将其所在的行与列清零。

```
示例 1:
输入:
[
  [1,1,1],
  [1,0,1],
  [1,1,1]
]
输出:
[
  [1,0,1],
  [0,0,0],
  [1,0,1]
]

示例 2:
输入:
[
  [0,1,2,0],
  [3,4,5,2],
  [1,3,1,5]
]
输出:
[
  [0,0,0,0],
  [0,4,5,0],
  [0,3,1,0]
]
```

2. 简单实现

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        int m = matrix.size();
        if(m == 0) return;
        int n = matrix[0].size();
        if(n == 0) return;
        unordered_set<int> row, col;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++){
                if(matrix[i][j] == 0){
                    row.insert(i);
                    col.insert(j);
                }
            }
        for(auto it = row.begin(); it != row.end(); it++){
            for(int j = 0; j < n; j++){
                matrix[*it][j] = 0;
            }
        }
        for(auto it = col.begin(); it != col.end(); it++){
            for(int i = 0; i < m; i++){
                matrix[i][*it] = 0;
            }
        }
    }
};
```

3. 原地算法

   - 要设零的行i：将该行的第一列元素 `matrix[i][0]` 设为0，表示该行需要清零；
   - 需要设为零的列j：将该列的第一行元素 `matrix[0][j]` 设为0，表示该列需要清零；
   - 根据第一行和第一列的标记，进行清零操作。
   - `matrix[0][0]` 即表示第一行又表示第一列,需要特殊处理

面试题01.09 字符串轮转（简单）

1. 题目描述

   字符串轮转。给定两个字符串s1和s2，请编写代码检查s2是否为s1旋转而成（比如，waterbottle是erbottlewat旋转后的字符串）。

```
示例1:
 输入：s1 = "waterbottle", s2 = "erbottlewat"
 输出：True

示例2:
 输入：s1 = "aa", "aba"
 输出：False
```

   提示字符串长：度在[0, 100000]范围内。 说明：你能只调用一次检查子串的方法吗?

2. 简单实现

依次从索引为0...len-2处旋转比较

```
class Solution {
public:
    bool isFlipedString(string s1, string s2) {
        if(s1.size() != s2.size()) return false;
        if(s1 == s2) return true;
        int len = s1.size();
        for(int i = 0; i < len-1; i++){
            if(s1[i] == s2[len-1]){//只有旋转点与s2最后一个字符相等时，才有旋转检查的必要
                string temp = s1.substr(i+1, len-i-1) + s1.substr(0, i+1);//旋转后字
符串
                if(temp == s2) return true;
            }
        }
        return false;
    }
};
```

3. 最优解法

如果成立，则s2必然在s1+s1里

```
class Solution {
    public boolean isFlipedString(String s1, String s2) {
        if(s1.length()!=s2.length())return false;
        if(s1.equals(s2))return true;
        s1+=s1;
        return s1.contains(s2);
    }
}
```

面试题02.01 移除重复节点（简单）

1. 题目描述

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

```
示例1:
 输入：[1, 2, 3, 3, 2, 1]
 输出：[1, 2, 3]

示例2:
 输入：[1, 1, 1, 1, 2]
 输出：[1, 2]
提示:
链表长度在[0, 20000]范围内。
链表元素在[0, 20000]范围内。
```

进阶：不得使用临时缓冲区，该怎么解决？

2. 简单实现

```
class Solution {
public:
    ListNode* removeDuplicateNodes(ListNode* head) {
        if(!head) return head;
        unordered_set<int> s;//借助集合记录出现过的值
        ListNode* pre = head;
        s.insert(head->val);
        ListNode* cur = head->next;
        while(cur){
            if(s.count(cur->val) > 0){
                pre->next = cur->next;
                cur = cur->next;
            }
            else{
                s.insert(cur->val);
                pre = cur;
                cur = cur->next;
            }
        }
        return head;
    }
};
```

3. 进阶实现

不能使用缓存，那就用暴力方法，每次遍历到一个节点，都从头遍历一遍看一看有没有重复值，如果有则抛弃

```
class Solution {
public:
    ListNode* removeDuplicateNodes(ListNode* head) {
        if(!head) return head;
        ListNode* pre = head;
        ListNode* cur = head->next;
        while(cur){
            ListNode* temp = head;//从头遍历
            bool dup = false;
            while(temp != cur){
                if(temp->val == cur->val){//出现重复值
                    dup = true;
                    break;
                }
                temp = temp->next;
            }
            if(dup){
                pre->next = cur->next;
                cur = cur->next;
            }
            else{
                pre = cur;
                cur = cur->next;
```

```
            }
        }
        return head;
    }
};
```

面试题02.02 倒数第k个节点（简单）

1. 题目描述

   实现一种算法，找出单向链表中倒数第 k 个节点。返回该节点的值。

   注意：本题相对原题稍作改动

   ```
   示例:
   输入： 1->2->3->4->5 和 k = 2
   输出： 4
   ```

   说明：给定的 k 保证是有效的。

2. 简单实现

```
class Solution {
public:
    int kthToLast(ListNode* head, int k) {
        ListNode* fast = head;
        while(--k)
            fast = fast->next;
        ListNode* slow = head;
        while(fast->next){
            fast = fast->next;
            slow = slow->next;
        }
        return slow->val;
    }
};
```

面试题02.03 删除中间节点（简单）

1. 题目描述

   实现一种算法，删除单向链表中间的某个节点（除了第一个和最后一个节点，不一定是中间节点），假定你只能访问该节点。

   ```
   示例:
   输入：单向链表a->b->c->d->e->f中的节点c
   结果：不返回任何数据，但该链表变为a->b->d->e->f
   ```

2. 简单实现

```
class Solution {
public:
    void deleteNode(ListNode* node) {
        node->val = node->next->val;
        node->next = node->next->next;
    }
};
```

面试题02.04 分割链表（中等）

1. 题目描述

   编写程序以 x 为基准分割链表，使得所有小于 x 的节点排在大于或等于 x 的节点之前。如果链表中包含 x，x 只需出现在小于 x 的元素之后(如下所示)。分割元素 x 只需处于“右半部分”即可，其不需要被置于左右两部分之间。

   ```
   示例:
   输入: head = 3->5->8->5->10->2->1, x = 5
   输出: 3->1->2->10->5->5->8
   ```

2. 简单实现

   按大小分成左右两部分，其实和按奇偶性分为两部分等题目是一样的

```
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode* hpart1 = new ListNode(-1);//左半部分的虚拟头节点
        ListNode* hpart2 = new ListNode(-1);//右半部分的虚拟头节点
        ListNode* p1 = hpart1;
        ListNode* p2 = hpart2;
        ListNode* cur = head;
        while(cur){
            if(cur->val < x){
                p1->next = cur;
                p1 = cur;
            }
            else{
                p2->next = cur;
                p2 = cur;
            }
            cur = cur->next;
        }
        p1->next = hpart2->next;//左右相连
        p2->next = NULL;//不能忘记右半部分尾节点置空节点，否则有可能形成循环链表
        return hpart1->next;
    }
};
```

面试题02.05 链表求和（中等）

1. 题目描述

给定两个用链表表示的整数，每个节点包含一个数位。这些数位是反向存放的，也就是个位排在链表首部。

编写函数对这两个整数求和，并用链表形式返回结果。

```
示例：
输入：(7 -> 1 -> 6) + (5 -> 9 -> 2)，即617 + 295
输出：2 -> 1 -> 9，即912
```

进阶：假设这些数位是正向存放的，请再做一遍。

```
示例：
输入：(6 -> 1 -> 7) + (2 -> 9 -> 5)，即617 + 295
输出：9 -> 1 -> 2，即912
```

2. 简单实现

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* ans = new ListNode(-1);
        ListNode* cur = ans;
        int c = 0;
        while(l1 && l2){
            cur->next = new ListNode(l1->val + l2->val + c);
            cur = cur->next;
            if(cur->val >= 10){
                cur->val -= 10;
                c = 1;
            }
            else c = 0;
            l1 = l1->next;
            l2 = l2->next;
        }
        if(l1){
            while(c && l1){
                cur->next = new ListNode(l1->val + c);
                cur = cur->next;
                if(cur->val >= 10){
                    cur->val -= 10;
                    c = 1;
                }
                else c = 0;
                l1 = l1->next;
            }
            if(l1) cur->next = l1;
        }
        if(l2){
            while(c && l2){
                cur->next = new ListNode(l2->val + c);
                cur = cur->next;
```

```
                if(cur->val >= 10){
                    cur->val -= 10;
                    c = 1;
                }
                else c = 0;
                l2 = l2->next;
            }
            if(l2) cur->next = l2;
        }
        if(c == 1) cur->next = new ListNode(1);
        return ans->next;
    }
};
```

PS：对于正向的，可以先反转链表按反向算，再将结果翻转即可

面试题02.06 回文链表（简单）

1. 题目描述

   编写一个函数，检查输入的链表是否是回文的。

```
示例 1:
输入:  1->2
输出:  false

示例 2:
输入:  1->2->2->1
输出:  true
```

   进阶：你能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题?

2. 简单实现

   之前做过，非O(1)空间要求的很简单，若O(1)，则要先将前半部分链表反转，然后对前半部分和后半部分同时遍历比较是否相等

面试题02.07 链表相交（简单）

1. 题目描述

   给定两个（单向）链表，判定它们是否相交并返回交点。请注意相交的定义基于节点的引用，而不是基于节点的值。换句话说，如果一个链表的第k个节点与另一个链表的第j个节点是同一节点（引用完全相同），则这两个链表相交。

```
示例 1:
输入: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB
= 3
输出: Reference of the node with value = 8
输入解释: 相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为
[4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3
```

个节点。

示例 2:
输入: intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
输出: Reference of the node with value = 2
输入解释: 相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；在 B 中，相交节点前有 1 个节点。

示例 3:
输入: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
输出: null
输入解释: 从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
解释: 这两个链表不相交，因此返回 null。

注意：

- 如果两个链表没有交点，返回 null 。
- 在返回结果后，两个链表仍须保持原有的结构。
- 可假定整个链表结构中没有循环。
- 程序尽量满足 O(n) 时间复杂度，且仅用 O(1) 内存。

2. 简单实现

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        int len1 = 0, len2 = 0;
        ListNode* p1 = headA;
        while(p1){
            len1++;
            p1 = p1->next;
        }
        ListNode* p2 = headB;
        while(p2){
            len2++;
            p2 = p2->next;
        }
        p1 = headA;
        p2 = headB;
        while(len1 > len2){
            p1 = p1->next;
            len1--;
        }
        while(len1 < len2){
            p2 = p2->next;
            len2--;
        }
        while(p1){
            if(p1 == p2) return p1;
            p1 = p1->next;
            p2 = p2->next;
        }
```

```
            return NULL;
        }
    };
```

3. 其他方法

    c++ 双指针， 把两个链表在逻辑上分别前后连起来变成两个等长的链表，同时不断向后移动，直到找到相同地址即相交点（最多遍历一圈半）或到空指针时停下来返回即可。

```
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        if(headA==nullptr || headB==nullptr) return NULL;
        ListNode *a = headA;
        ListNode *b = headB;
        while(a!=b){
            if(!a) a = headB;//逻辑连接
            else a = a->next;
            if(!b) b = headA;
            else b = b->next;//逻辑连接
        }
        return a;
    }
};
```

面试题02.08 环路检测（中等）

1. 题目描述

    给定一个有环链表，实现一个算法返回环路的开头节点。 有环链表的定义：在链表中某个节点的next元素指向在它前面出现过的节点，则表明该链表存在环路。

```
示例 1:
输入: head = [3,2,0,-4], pos = 1
输出: tail connects to node index 1
解释: 链表中有一个环，其尾部连接到第二个节点。

示例 2:
输入: head = [1,2], pos = 0
输出: tail connects to node index 0
解释: 链表中有一个环，其尾部连接到第一个节点。

示例 3:
输入: head = [1], pos = -1
输出: no cycle
解释: 链表中没有环。
```

    进阶：你是否可以不用额外空间解决此题?

2. 简单实现

```
```

```
class Solution {
public:
    struct ListNode *detectCycle(struct ListNode *head) {
        struct ListNode *slow = head, *fast = head;
        // 快慢指针
        while (fast != NULL && fast->next != NULL){
            slow = slow->next;
            fast = fast->next->next;
            // 如果相遇了就break
            if (slow == fast) break;
        }
        // fast到了链表尾部,说明链表无环
        if (fast == NULL || fast->next == NULL) return NULL;
        // 慢指针从头开始，快慢指针再一次相遇就是在环的起点
        slow = head;
        while (slow != fast){
            slow = slow->next;
            fast = fast->next;
        }
        return fast;
    }
};
```

面试题03.01 三合一（中等）

1. 题目描述

   三合一。描述如何只用一个数组来实现三个栈。

   你应该实现push(stackNum, value)、pop(stackNum)、isEmpty(stackNum)、peek(stackNum)方法。stackNum表示栈下标，value表示压入的值。

   构造函数会传入一个stackSize参数，代表每个栈的大小。

```
示例1:
 输入:
"TripleInOne", "push", "push", "pop", "pop", "pop", "isEmpty", [0, 1], [0, 2], [0],
[0], [0], [0]]
 输出:
[null, null, null, 1, -1, -1, true]
说明：当栈为空时pop，peek返回-1，当栈满时push不压入元素。

示例2:
 输入:
"TripleInOne", "push", "push", "push", "pop", "pop", "pop", "peek", [0, 1], [0, 2],
[0, 3], [0], [0], [0], [0]]
 输出:
[null, null, null, null, 2, 1, -1, -1]
```

2. 简单实现

   既然会给出栈的大小stackSize，直接构造大小为3*（1+stackSize）的数组存储即可，其中每个栈多出一位标识当前栈内元素个数的位置

```
class TripleInOne {
public:
    vector<int> data;
    int size;
    TripleInOne(int stackSize) {
        size = stackSize;
        data = vector<int>((1+size)*3);
        data[0*(size+1)] = 0;//仅初始化三个标识位即可，省时间
        data[1*(size+1)] = 0;
        data[2*(size+1)] = 0;
    }
    void push(int stackNum, int value) {
        if(data[stackNum*(size+1)] < size){//该栈未满
            int idx = stackNum*(size+1) + data[stackNum*(size+1)] + 1;//将要push的
idx
            data[idx] = value;
            data[stackNum*(size+1)]++;
        }
    }
    int pop(int stackNum) {
        if(data[stackNum*(size+1)] > 0){
            int idx = stackNum*(size+1) + data[stackNum*(size+1)];
            data[stackNum*(size+1)]--;
            return data[idx];
        }
        else return -1;
    }
    int peek(int stackNum) {
        if(data[stackNum*(size+1)] > 0){
            int idx = stackNum*(size+1) + data[stackNum*(size+1)];
            return data[idx];
        }
        else return -1;
    }
    bool isEmpty(int stackNum) {
        return data[stackNum*(size+1)] == 0;
    }
};
```

面试题03.02 栈的最小值（简单）

1. 题目描述

   请设计一个栈，除了常规栈支持的pop与push函数以外，还支持min函数，该函数返回栈元素中的最小值。执行push、pop和min操作的时间复杂度必须为O(1)。

```
示例:
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

2. 简单实现

   再用一个栈记录最小值即可

```
class MinStack {
public:
    stack<int> data;
    stack<int> minn;
    /** initialize your data structure here. */
    MinStack() {
    }
    void push(int x) {
        data.push(x);
        if(minn.empty() || minn.top() >= x)
            minn.push(x);
    }
    void pop() {
        if(!minn.empty() && data.top() == minn.top())
            minn.pop();
        data.pop();
    }
    int top() {
        return data.top();
    }
    int getMin() {
        return minn.top();
    }
};
```

面试题03.03 堆盘子（中等）

1. 题目描述

   堆盘子。设想有一堆盘子，堆太高可能会倒下来。因此，在现实生活中，盘子堆到一定高度时，我们就会另外堆一堆盘子。请实现数据结构SetOfStacks，模拟这种行为。SetOfStacks应该由多个栈组成，并且在前一个栈填满时新建一个栈。此外，SetOfStacks.push()和SetOfStacks.pop()应该与普通栈的操作方法相同（也就是说，pop()返回的值，应该跟只有一个栈时的情况一样）。 进阶：实现一个popAt(int index)方法，根据指定的子栈，执行pop操作。

   当某个栈为空时，应当删除该栈。当栈中没有元素或不存在该栈时，pop，popAt 应返回 -1.

```
示例1:
 输入:
"StackOfPlates", "push", "push", "popAt", "pop", "pop", [1], [2], [1], [], []]
 输出:
[null, null, null, 2, 1, -1]

示例2:
 输入:
"StackOfPlates", "push", "push", "push", "popAt", "popAt", "popAt", [1], [2], [3],
[0], [0], [0]]
 输出:
[null, null, null, null, 2, 1, 3]
```

2. 简单实现

```
class StackOfPlates {
public:
    vector<stack<int>> v;
    int cap;
    int ridx;//当前最后一个栈的idx
    StackOfPlates(int cap) {
        this->cap = cap;
        ridx=-1;
    }
    void push(int val) {
        if(cap == 0) return;//防止cap=0时出错
        if(ridx == -1 || v[ridx].size() == cap){
            stack<int> cur;
            cur.push(val);
            v.push_back(cur);
            ridx++;
        }
        else v[ridx].push(val);
    }

    int pop() {
        if(cap == 0) return -1;
        if(ridx == -1) return -1;
        int ans = v[ridx].top();
        if(v[ridx].size() > 1)
            v[ridx].pop();
        else{
            v.erase(v.begin()+ridx);
            ridx--;
        }
        return ans;
    }

    int popAt(int index) {
        if(cap == 0) return -1;
        if(index > ridx) return -1;
        int ans = v[index].top();
```

```
            if(v[index].size() > 1)
                v[index].pop();
            else{
                v.erase(v.begin()+index);
                ridx--;
            }
            return ans;
        }
    };
```

面试题03.04 化栈为队（简单）

1. 题目描述

   实现一个MyQueue类，该类用两个栈来实现一个队列。

   ```
   示例:
   MyQueue queue = new MyQueue();
   queue.push(1);
   queue.push(2);
   queue.peek();  // 返回 1
   queue.pop();   // 返回 1
   queue.empty(); // 返回 false
   ```

   说明:

   - 你只能使用标准的栈操作 -- 也就是只有 push to top, peek/pop from top, size 和 is empty 操作是合法的。
   - 你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。
   - 假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。

2. 简单实现

   两个栈来回倒，效率并不高的样子

   ```
   class MyQueue {
   public:
       stack<int> s1;
       stack<int> s2;
       /** Initialize your data structure here. */
       MyQueue() {
       }
       /** Push element x to the back of queue. */
       void push(int x) {
           while(!s2.empty()){
               s1.push(s2.top());
               s2.pop();
           }
           s1.push(x);
       }
       /** Removes the element from in front of queue and returns that element. */
       int pop() {
   ```

```
        while(!s1.empty()){
            s2.push(s1.top());
            s1.pop();
        }
        int ans = s2.top();
        s2.pop();
        return ans;
    }
    /** Get the front element. */
    int peek() {
        while(!s1.empty()){
            s2.push(s1.top());
            s1.pop();
        }
        return s2.top();
    }
    /** Returns whether the queue is empty. */
    bool empty() {
        return s1.empty() && s2.empty();
    }
};
```

面试题03.05 栈排序（中等）

1. 题目描述

   栈排序。 编写程序，对栈进行排序使最小元素位于栈顶。最多只能使用一个其他的临时栈存放数据，但不得将元素复制到别的数据结构（如数组）中。该栈支持如下操作：push、pop、peek 和 isEmpty。当栈为空时，peek 返回 -1。

```
示例1:
 输入:
"SortedStack", "push", "push", "peek", "pop", "peek", [1], [2], [], [], []]
 输出:
[null,null,null,1,null,2]

示例2:
 输入:
"SortedStack", "pop", "pop", "push", "pop", "isEmpty", [], [], [1], [], []]
 输出:
[null,null,null,null,null,true]
```

   说明: 栈中的元素数目在[0, 5000]范围内。

2. 简单实现

   实际上要求栈内从顶到底按从小到大的顺序排序，因此在插入时保证顺序即可

```
class SortedStack {
public:
    stack<int> data;
    stack<int> cache;
```

```
    SortedStack() {
    }
    void push(int val) {
        if(data.empty() || data.top() >= val)
            data.push(val);
        else{
            while(!data.empty() && data.top() < val){
                cache.push(data.top());
                data.pop();
            }
            data.push(val);
            while(!cache.empty()){
                data.push(cache.top());
                cache.pop();
            }
        }
    }
    void pop() {
        if(!data.empty())
            data.pop();
    }
    int peek() {
        if(!data.empty())
            return data.top();
        else
            return -1;
    }
    bool isEmpty() {
        return data.empty();
    }
};
```

PS：可以改进一下，不用非得始终将数据维护在data里，只要动态维护data和cache即可，可以减少一些出入栈的次数

面试题03.06 动物收容所（简单）

1. 题目描述

动物收容所。有家动物收容所只收容狗与猫，且严格遵守“先进先出”的原则。在收养该收容所的动物时，收养人只能收养所有动物中“最老”（由其进入收容所的时间长短而定）的动物，或者可以挑选猫或狗（同时必须收养此类动物中“最老”的）。换言之，收养人不能自由挑选想收养的对象。请创建适用于这个系统的数据结构，实现各种操作方法，比如enqueue、dequeueAny、dequeueDog和dequeueCat。允许使用Java内置的LinkedList数据结构。

enqueue方法有一个animal参数，animal[0]代表动物编号，animal[1]代表动物种类，其中 0 代表猫，1 代表狗。

dequeue*方法返回一个列表[动物编号, 动物种类]，若没有可以收养的动物，则返回[-1,-1]。

```
示例1:
 输入:
"AnimalShelf", "enqueue", "enqueue", "dequeueCat", "dequeueDog", "dequeueAny", [[0,
0]], [[1, 0]], [], [], []]
 输出:
[null,null,null,[0,0],[-1,-1],[1,0]]

示例2:
 输入:
"AnimalShelf", "enqueue", "enqueue", "enqueue", "dequeueDog", "dequeueCat",
"dequeueAny", [[0, 0]], [[1, 0]], [[2, 1]], [], [], []]
 输出:
[null,null,null,null,[2,1],[0,0],[1,0]]
```

说明: 收纳所的最大容量为20000

2. 简单实现

各自维护一个对列，并在入队时记录时间戳

```
class AnimalShelf {
public:
    queue<vector<int>> cat;
    queue<vector<int>> dog;
    int t;//时间戳
    AnimalShelf() {
        t = 0;
    }
    void enqueue(vector<int> animal) {
        if(animal[1] == 0)
            cat.push({animal[0], 0, t});
        else
            dog.push({animal[0], 1, t});
        t++;
    }
    vector<int> dequeueAny() {
        vector<int> ans;
        if(cat.empty() && dog.empty())
            return {-1,-1};
        else if(cat.empty()){
            ans = {dog.front()[0], 1};
            dog.pop();
        }
        else if(dog.empty()){
            ans = {cat.front()[0], 0};
            cat.pop();
        }
        else if(cat.front()[2] < dog.front()[2]){
            ans = {cat.front()[0], 0};
            cat.pop();
        }
        else{
            ans = {dog.front()[0], 1};
```

```
            dog.pop();
        }
        return ans;
    }
    vector<int> dequeueDog() {
        if(dog.empty())
            return {-1, -1};
        vector<int> ans = {dog.front()[0], 1};
        dog.pop();
        return ans;
    }
    vector<int> dequeueCat() {
        if(cat.empty())
            return {-1, -1};
        vector<int> ans = {cat.front()[0], 0};
        cat.pop();
        return ans;
    }
};
```

面试题04.01 节点间通路（中等）

1. 题目描述

   节点间通路。给定有向图，设计一个算法，找出两个节点之间是否存在一条路径。

```
示例1:
 输入：n = 3, graph = [[0, 1], [0, 2], [1, 2], [1, 2]], start = 0, target = 2
 输出：true

示例2:
 输入：n = 5, graph = [[0, 1], [0, 2], [0, 4], [0, 4], [0, 1], [1, 3], [1, 4], [1,
3], [2, 3], [3, 4]], start = 0, target = 4
 输出 true
```

   提示：

   - 节点数量n在[0, 1e5]范围内。
   - 节点编号大于等于 0 小于 n。
   - 图中可能存在自环和平行边。

2. 简单实现——BFS

```
```

面试题04.02 最小高度树（简单）

1. 题目描述

   给定一个有序整数数组，元素各不相同且按升序排列，编写一个算法，创建一棵高度最小的二叉搜索树。

```
示例：
给定有序数组：[-10,-3,0,5,9],

一个可能的答案是：[0,-3,9,-10,null,5]，它可以表示下面这个高度平衡二叉搜索树：
          0
     / \
   -3   9
   /   /
 -10  5
```

2. 简单实现

```
class Solution {
public:
    TreeNode* ToBST(vector<int>& nums, int l, int r) {
        if(l > r) return NULL;
        if(l == r) return new TreeNode(nums[l]);
        int mid = l + (r-l)/2;
        TreeNode* root = new TreeNode(nums[mid]);
        root->left = ToBST(nums, l, mid-1);
        root->right = ToBST(nums, mid+1, r);
        return root;
    }
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return ToBST(nums, 0, nums.size()-1);
    }
};
```

面试题04.03 特定深度节点链表（中等）

1. 题目描述

给定一棵二叉树，设计一个算法，创建含有某一深度上所有节点的链表（比如，若一棵树的深度为 D，则会创建出 D 个链表）。返回一个包含所有深度的链表的数组。

```
示例：
输入：[1,2,3,4,5,null,7,8]
                1
        /  \
       2    3
      / \    \
     4   5    7
    /
   8
输出：[[1],[2,3],[4,5,7],[8]]
```

2. 简单实现

```
class Solution {
public:
```

```
    vector<ListNode*> listOfDepth(TreeNode* tree) {
        if(!tree) return {};
        vector<ListNode*> ans;
        queue<TreeNode*> q;
        q.push(tree);
        ListNode* head = new ListNode(-1);
        while(!q.empty()){
            int size = q.size();
            ListNode* cur = head;
            for(int i = 0; i < size; i++){
                TreeNode* temp = q.front();
                q.pop();
                cur->next = new ListNode(temp->val);
                cur = cur->next;
                if(temp->left) q.push(temp->left);
                if(temp->right) q.push(temp->right);
            }
            ans.push_back(head->next);
        }
        return ans;
    }
};
```

面试题04.04 检查平衡性（简单）

1. 题目描述

   实现一个函数，检查二叉树是否平衡。在这个问题中，平衡树的定义如下：任意一个节点，其两棵子树的高度差不超过 1。

```
示例 1:
给定二叉树 [3,9,20,null,null,15,7]
    3
   / \
  9  20
    /  \
   15   7
返回 true 。

示例 2:
给定二叉树 [1,2,2,3,3,null,null,4,4]
      1
     / \
    2   2
   / \
  3   3
 / \
4   4
返回 false 。
```

2. 简单实现

```
class Solution {
public:
    int Get_Deep(TreeNode *root) {
        if(root==NULL)
            return 0;
        return max(Get_Deep(root->left),Get_Deep(root->right))+1;
    }
    bool isBalanced(TreeNode* root) {
        if(root==NULL)
            return true;
        int left_deep=Get_Deep(root->left);
        int right_depp=Get_Deep(root->right);
        return isBalanced(root->left)&&isBalanced(root->right)&&(abs(right_depp-
left_deep)<=1);
    }
};
```

面试题04.05 合法二叉搜索树（中等）

1. 题目描述

   实现一个函数，检查一棵二叉树是否为二叉搜索树。

```
示例 1:
输入:
    2
   / \
  1   3
输出: true

示例 2:
输入:
    5
   / \
  1   4
     / \
    3   6
输出: false
解释: 输入为: [5,1,4,null,null,3,6]。
     根节点的值为 5 ，但是其右子节点值为 4 。
```

2. 简单实现

```
class Solution {
public:
    long pre = long(INT_MIN)-1;
    bool ans = true;
    bool isValidBST(TreeNode* root) {
        if(!ans) return ans;
        if(!root) return ans;
        ans = isValidBST(root->left);
```

```
            if(!ans) return ans;
            if(root->val > pre)
                pre = root->val;
            else{
                ans = false;
                return false;
            }
            ans = isValidBST(root->right);
            return ans;
        }
    };
```

面试题04.06 后继者（中等）

1. 题目描述

   设计一个算法，找出二叉搜索树中指定节点的“下一个”节点（也即中序后继）。

   如果指定节点没有对应的“下一个”节点，则返回null。

```
示例 1:
输入: root = [2,1,3], p = 1
  2
 / \
1   3
输出: 2

示例 2:
输入: root = [5,3,6,2,4,null,null,1], p = 6
  5
 / \
3   6
/ \
2   4
/
1
输出: null
```

2. 简单实现

```
class Solution {
public:
    TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
        if(!p) return NULL;
        if(p->right){//后继结点在右子树上
            TreeNode* cur = p->right;
            while(cur->left)
                cur = cur->left;
            return cur;
        }
```

```
        else{//后继结点在从根节点到p的路径上或不存在
            TreeNode* cur = root;//从根节点开始寻找p
            TreeNode* ans = NULL;//记录路径上最新的值大于p的节点，实际上就是p的后继结点
            while(cur && cur->left != p && cur->right != p){
                if(cur->val > p->val)
                    ans = cur;
                if(cur->val > p->val)
                    cur = cur->left;
                else
                    cur = cur->right;
            }
            if(cur && cur->val > p->val)//cur可能为NULL，代表给的P不在树上。。。
                ans = cur;
            return ans;
        }
    }
};
```

3. 递归解法

所谓 p 的后继节点，就是这串升序数字中，比 p 大的下一个。

- 如果 p 大于当前节点的值，说明后继节点一定在 RightTree
- 如果 p 等于当前节点的值，说明后继节点一定在 RightTree
- 如果 p 小于当前节点的值，说明后继节点一定在 LeftTree 或自己就是
    - 递归调用 LeftTree，如果是空的，说明当前节点就是答案
    - 如果不是空的，则说明在 LeftTree 已经找到合适的答案，直接返回即可

```
TreeNode* inorderSuccessor(TreeNode* root, TreeNode* p) {
        if (!root) {
            return NULL;
        }

        if (root->val <= p->val) {
            return inorderSuccessor(root->right, p);
        } else {
            TreeNode *leftRet = inorderSuccessor(root->left, p);
            if (!leftRet) {
                return root;
            } else {
                return leftRet;
            }
        }
    }
```

面试题04.08 首个共同祖先（中等）

1. 题目描述

   设计并实现一个算法，找出二叉树中某两个节点的第一个共同祖先。不得将其他的节点存储在另外的数据结构中。注意：这不一定是二叉搜索树。

```
例如，给定如下二叉树: root = [3,5,1,6,2,0,8,null,null,7,4]
    3
  / \
  5   1
 / \ / \
6  2 0  8
  / \
 7   4
示例 1:
输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
输入: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。
示例 2:
输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
输出: 5
解释: 节点 5 和节点 4 的最近公共祖先是节点 5。因为根据定义最近公共祖先节点可以为节点本身。
```

   说明:

   - 所有节点的值都是唯一的
   - p、q 为不同节点且均存在于给定的二叉树中。

2. 简单实现

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root == NULL) //两个节点均不属于这棵子树
            return NULL;
        if(p == root || q == root) // p属于这个子树，或者q属于这个子树
            return root;
        TreeNode* left = lowestCommonAncestor(root->left, p, q); //判断p q 是否同属于左子树
        TreeNode* right = lowestCommonAncestor(root->right, p, q); //判断p q 是否同属于右子树
        if(left != NULL && right != NULL) // 说明一个属于左子树，一个属于右子树
            return root; //则返回根节点
        return left != NULL ? left : right; // 如果哪个子树不为NULL则两个节点均属于那个子树
    }
};
```

面试题04.09 二叉搜索树序列（困难）

1. 题目描述

从左向右遍历一个数组，通过不断将其中的元素插入树中可以逐步地生成一棵二叉搜索树。给定一个由不同节点组成的二叉树，输出所有可能生成此树的数组。

```
示例:
给定如下二叉树
    2
   / \
  1   3
  返回:
[
   [2,1,3],
   [2,3,1]
]
```

2. 简单实现

只要保证所有的父节点都出现在其子节点之前即可，因此可以看作拓扑排序问题，把树看作拓扑图，列出所有可能的拓扑排序结果

```
class Solution {
public:
    vector<vector<int>> BSTSequences(TreeNode* root) {
        if(!root) return {{}};
        vector<vector<int>> v1 = {{root->val}};//保存目前所有已拓扑排序的节点结果
        vector<vector<TreeNode*>> t1;//保存下一次可以选择的节点，与t1[i]与v1[i]相对应
        if(root->left && root->right)
            t1.push_back({root->left, root->right});
        else if(root->left)
            t1.push_back({root->left});
        else if(root->right)
            t1.push_back({root->right});
        else//只有一个根节点，可以直接返回
            return v1;
        vector<vector<int>> v2;
        vector<vector<TreeNode*>> t2;
        while(t1[0].size() > 0){//无可选择节点说明排序完毕
            for(int i = 0; i < v1.size(); i++){//遍历每一个已排序序列
                for(int j = 0; j < t1[i].size(); j++){//依次遍历可选节点
                    vector<int> cur = v1[i];
                    cur.push_back(t1[i][j]->val);//排入节点
                    v2.push_back(cur);
                    vector<TreeNode*> temp = t1[i];
                    //将新排入节点的孩子节点加入下一次遍历的可选节点中，注意去掉已经排入的
t1[i][j]
                    if(t1[i][j]->left && t1[i][j]->right){
                        temp[j] = t1[i][j]->left;
                        temp.push_back(t1[i][j]->right);
                    }
                    else if(t1[i][j]->left)
                        temp[j] = t1[i][j]->left;
                    else if(t1[i][j]->right)
```

```
                        temp[j] = t1[i][j]->right;
                    else
                        temp.erase(temp.begin() + j);
                    t2.push_back(temp);
                }
            }
            v1 = v2;
            t1 = t2;
            v2.clear();
            t2.clear();
        }
        return v1;
    }
};
```

3. 递归

搜索

- 使用一个queue存储下个所有可能的节点
- 然后选择其中一个作为path的下一个元素
- 递归直到queue元素为空
- 将对应的path加入结果中
- 由于二叉搜索树没有重复元素, 而且每次递归的使用元素的顺序都不一样, 所以自动做到了去重

```
class Solution:
    def BSTSequences(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return [[]]
        res = []
        def findPath(cur, q, path):
            if cur.left:
                q.append(cur.left)
            if cur.right:
                q.append(cur.right)
            if not q:
                res.append(path)
                return
            for i, nex in enumerate(q):
                newq = q[:i] + q[i + 1:]
                findPath(nex, newq, path + [nex.val])
        findPath(root, [], [root.val])
        return res
```

面试题04.10 检查子树（中等）

1. 题目描述

检查子树。你有两棵非常大的二叉树：T1，有几万个节点；T2，有几万个节点。设计一个算法，判断 T2 是否为 T1 的子树。

如果 T1 有这么一个节点 n，其子树与 T2 一模一样，则 T2 为 T1 的子树，也就是说，从节点 n 处把树砍断，得到的树与 T2 完全相同。

```
示例1:
 输入：t1 = [1, 2, 3], t2 = [2]
 输出：true

示例2:
 输入：t1 = [1, null, 2, 4], t2 = [3, 2]
 输出：false
```

提示：树的节点数目范围为[0, 20000]。

2. 简单实现

a

```
class Solution {
public:
    bool cmp(TreeNode* t1, TreeNode* t2){//比较两棵树是否相等
        if(!t1 && !t2) return true;
        if(!t1 || !t2) return false;
        if(t1->val != t2->val) return false;
        return cmp(t1->left, t2->left) && cmp(t1->right, t2->right);
    }
    bool ans = false;
    bool checkSubTree(TreeNode* t1, TreeNode* t2) {//先序遍历搜索t1
        if(ans) return ans;//找到了就提前return停止处理
        if(!t2) return true;
        if(!t1) return false;
        if(cmp(t1, t2))
            ans = true;
        else
            ans = checkSubTree(t1->left, t2) || checkSubTree(t1->right, t2);
        return ans;
    }
};
```

面试题04.12 求和路径（中等）

1. 题目描述

给定一棵二叉树，其中每个节点都含有一个整数数值(该值或正或负)。设计一个算法，打印节点数值总和等于某个给定值的所有路径的数量。注意，路径不一定非得从二叉树的根节点或叶节点开始或结束，但是其方向必须向下(只能从父节点指向子节点方向)。

```
示例:
给定如下二叉树，以及目标和 sum = 22,
                5
          / \
         4   8
        /   / \
       11  13  4
      /  \    / \
     7    2  5   1
返回:3
解释: 和为 22 的路径有: [5,4,11,2], [5,8,4,5], [4,11,7]
```

提示：节点总数 <= 10000

2. 简单实现

先序遍历+DFS

```
class Solution {
public:
    int ans = 0;
    void dfs(TreeNode* root, int& temp, const int& aim){
        if(!root) return;
        temp += root->val;
        if(temp == aim) ans++;//节点有正有负，所以要继续遍历下去
        dfs(root->left, temp, aim);
        dfs(root->right, temp, aim);
        temp -= root->val;
    }
    int pathSum(TreeNode* root, int sum) {//以root为起始点的和为sum的路径数
        if(!root) return 0;
        int temp = 0;
        dfs(root, temp, sum);//以根节点为起点DFS寻找和为sum的路径
        pathSum(root->left, sum);//以左子树为起点
        pathSum(root->right, sum);//以右子树为起点
        return ans;
    }
};
```

3. 前缀和方法

这道题用到了一个概念，叫前缀和。就是到达当前元素的路径上，之前所有元素的和。

前缀和怎么应用呢?

如果两个数的**前缀总和**是相同的，那么这些**节点之间的元素总和为零**。进一步扩展相同的想法，如果前缀总和currSum，在节点A和节点B处相差target，则位于节点A和节点B之间的元素之和是target。

因为本题中的路径是一棵树，从根往任一节点的路径上(不走回头路)，**有且仅有一条路径**，因为**不存在环**。(如果存在环，前缀和就不能用了，需要改造算法)

抵达当前节点(即B节点)后，将前缀和累加，然后查找在前缀和上，有没有**前缀和currSum-target的节点**(即A节点)，存在即表示从A到B有一条路径之和满足条件的情况。结果加上满足**前缀和currSum-target的节点**的数量。然后递归进入左右子树。

左右子树遍历完成之后，回到当前层，需要把当前节点添加的前缀和去除。避免回溯之后影响上一层。因为思想是前缀和，不属于前缀的，我们就要去掉它。

```
class Solution {
    public int pathSum(TreeNode root, int sum) {
        // key是前缀和, value是大小为key的前缀和出现的次数
        Map<Integer, Integer> prefixSumCount = new HashMap<>();
        // 前缀和为0的一条路径
        prefixSumCount.put(0, 1);
        // 前缀和的递归回溯思路
        return recursionPathSum(root, prefixSumCount, sum, 0);
    }
    /**
     * 前缀和的递归回溯思路
     * 从当前节点反推到根节点(反推比较好理解，正向其实也只有一条)，有且仅有一条路径，因为这是一棵
树
     * 如果此前有和为currSum-target,而当前的和又为currSum,两者的差就肯定为target了
     * 所以前缀和对于当前路径来说是唯一的，当前记录的前缀和，在回溯结束，回到本层时去除，保证其不
影响其他分支的结果
     * @param node 树节点
     * @param prefixSumCount 前缀和Map
     * @param target 目标值
     * @param currSum 当前路径和
     * @return 满足题意的解
     */
    private int recursionPathSum(TreeNode node, Map<Integer, Integer>
prefixSumCount, int target, int currSum) {
        // 1.递归终止条件
        if (node == null) {
            return 0;
        }
        // 2.本层要做的事情
        int res = 0;
        // 当前路径上的和
        currSum += node.val;

        //---核心代码
        // 看看root到当前节点这条路上是否存在节点前缀和加target为currSum的路径
```

```
        // 当前节点->root节点反推，有且仅有一条路径，如果此前有和为currSum-target,而当前的和
又为currSum,两者的差就肯定为target了
        // currSum-target相当于找路径的起点，起点的sum+target=currSum，当前点到起点的距离就
是target
        res += prefixSumCount.getOrDefault(currSum - target, 0);
        // 更新路径上当前节点前缀和的个数
        prefixSumCount.put(currSum, prefixSumCount.getOrDefault(currSum, 0) + 1);
        //---核心代码

        // 3.进入下一层
        res += recursionPathSum(node.left, prefixSumCount, target, currSum);
        res += recursionPathSum(node.right, prefixSumCount, target, currSum);

        // 4.回到本层，恢复状态，去除当前节点的前缀和数量
        prefixSumCount.put(currSum, prefixSumCount.get(currSum) - 1);
        return res;
    }
}
```

面试题05.01 插入（简单）

1. 题目描述

   插入。给定两个32位的整数N与M，以及表示比特位置的i与j。编写一种方法，将M插入N，使得M从N的第j位开始，到第i位结束。假定从j位到i位足以容纳M，也即若M = 10 011，那么j和i之间至少可容纳5个位。例如，不可能出现j = 3和i = 2的情况，因为第3位和第2位之间放不下M。

```
示例1:
 输入：N = 10000000000, M = 10011, i = 2, j = 6
 输出：N = 10001001100

示例2:
 输入：  N = 0, M = 11111, i = 0, j = 4
 输出：N = 11111
```

2. 简单实现

   以样例为例子：

   - 首先计算出一个和M一样长度的mask: 11111
   - 将mask向左移动i位，将N在移位后mask下的bit置0. N & (~(mask << i))
   - 将M左移i位，与N进行按位或。

```
class Solution {
public:
    int insertBits(int N, int M, int i, int j) {
        int len = j - i + 1;
        int mask = (1 << len) - 1;
        return (N & (~(mask << i))) | (M << i);
    }
};
```

面试题05.02 二进制数转字符串（中等）

1. 题目描述

   二进制数转字符串。给定一个介于0和1之间的实数（如0.72），类型为double，打印它的二进制表达式。如果该数字无法精确地用32位以内的二进制表示，则打印“ERROR”。

```
示例1:
 输入：0.625
 输出："0.101"

示例2:
 输入：0.1
 输出："ERROR"
 提示：0.1无法被二进制准确表示
```

   提示：32位包括输出中的"0."这两位。

2. 简单实现

   本来以为会涉及浮点数的精度问题，结果没有。。。

```
class Solution {
public:
    string printBin(double num) {
        string s = "0.";
        while(num != 0.0 && s.size() < 32){
            num *= 2;
            if(num >= 1.0){
                s += '1';
                num -= 1.0;
            }
            else
                s += '0';
        }
        if(num == 0.0)
            return s;
        else
            return "ERROR";
    }
};
```

面试题05.03 翻转数位（简单）

1. 题目描述

   给定一个32位整数 num，你可以将一个数位从0变为1。请编写一个程序，找出你能够获得的最长的一串1的长度。

```
示例 1:
输入: num = 1775(110111011112)
输出: 8

示例 2:
输入: num = 7(01112)
输出: 4
```

2. 简单实现

   暴力遍历即可解决，但我写了一个DP，结果双百

```
class Solution {
public:
    string tostr(int num){//转换为二进制字符串
        string s = "";
        while(num){
            s += num % 2 + '0';
            num /= 2;
        }
        if(s.size() < 32)
            s += '0';//处理题目中示例2的情况
        reverse(s.begin(), s.end());
        return s;
    }
    int reverseBits(int num) {
        string s = tostr(num);
        int len = s.size();
        vector<vector<int>> dp(len, vector<int>(2, 0));//以s[i]结尾的最长未替换过(0)和替换过(1)的串
        if(s[0] == '0'){
            dp[0][0] = 0;
            dp[0][1] = 1;
        }
        else{
            dp[0][0] = 1;
            dp[0][1] = 1;
        }
        int ans = 1;
        for(int i = 1; i < len; i++){
            if(s[i] == '0'){
                dp[i][0] = 0;
                dp[i][1] = dp[i-1][0] + 1;
            }
            else{
```

```
                dp[i][0] = dp[i-1][0] + 1;
                dp[i][1] = dp[i-1][1] + 1;
            }
            ans = max(max(ans, dp[i][0]), dp[i][1]);
        }
        return ans;
    }
};
```

面试题05.04 下一个数（中等）

1. 题目描述

   下一个数。给定一个正整数，找出与其二进制表达式中1的个数相同且大小最接近的那两个数（一个略大，一个略小）。

```
示例1:
 输入：num = 2（或者0b10）
 输出：[4, 1] 或者 ([0b100, 0b1])

示例2:
 输入：num = 1
 输出：[2, -1]
```

   提示:

   - num的范围在[1, 2147483647]之间;
   - 如果找不到前一个或者后一个满足条件的正数，那么输出 -1

2. 简单实现

```
class Solution {
public:
    int countOne(int num){//数二进制中1的个数
        int ans = 0;
        while(num){
            if(num % 2 == 1)
                ans++;
            num /= 2;
        }
        return ans;
    }
    vector<int> findClosedNumbers(int num) {
        if(num == 1) return {2, -1};//只有两个边界会有不满足条件的情况
        if(num == 2147483647) return {-1, -1};
        int cnt = countOne(num);
        int larger = num + 1;
        while(countOne(larger) != cnt) //向上找到略大的那个数
            larger++;
        int smaller = num - 1;
        while(countOne(smaller) != cnt) //向下找到略小的那个数
            smaller--;
```

```
        return {larger, smaller};
    }
};
```

3. 构造法

PS：注意bitset在处理二进制时的妙用

```
class Solution {
public:
    vector<int> findClosedNumbers(int num) {
        return { find_higher(num),find_lower(num) };
    }

    int find_lower(int num) {//找到略小的
        /*
        例如:   xxxxxxxxxxxxxxxxx101000000111111 找到第一个右端有0的1
        第一步: xxxxxxxxxxxxxxxxx100100000111111 交换这个1和右边的0
        第二步: xxxxxxxxxxxxxxxxx100111111100000 剩下的低位中的1尽可能左移
        完成!目标:找到第一个可以与右端交换的 1 ,之后将 i 位置右端的 1 尽量左移
        */
        bitset<32>tmp(num);//很方便地将num转换为可以处理的二进制
        //第一步
        int i = 1;
        while (i < 31 && !(tmp[i] && !tmp[i - 1]))++i;
        if (i == 31) return -1;//符号位置,无法操作
        tmp[i] = 0, tmp[i - 1] = 1;
        //第二步,注意此处是 i-2,因为 i-1 的位置已经是 1 了
        int j = i - 2;
        i = 0;
        while (j > i) {
            while(j > i && tmp[j]) j--;
            while(j > i && !tmp[i]) i++;
            if(j > i){
                tmp[j--] = 1;
                tmp[i++] = 0;
            }
        }
        return tmp.to_ulong();
    }

    int find_higher(int num) {
        /*
        例如:   xxxxxxxxxxxxxxxxx101000010111110 找到第一个左端有0的1
        第一步: xxxxxxxxxxxxxxxxx101000011011110 交换这个1和左边的0
        第二步: xxxxxxxxxxxxxxxxx101000011001111 剩下的低位中的1尽可能右移
        完成!目标:找到第一个可以与左端交换的 1 ,之后将  i 位置右端的 1 尽量右移
        */
        bitset<32>tmp(num);
        //第一步
        int i = 0;
        while (i < 31 && !(tmp[i] && !tmp[i + 1]))++i;
        if (i == 31) return -1;//符号位置,无法操作
```

```
            tmp[i] = 0, tmp[i + 1] = 1;
            //第二步
            int j = i - 1;
            i = 0;
            while (j > i) {
                while(j > i && !tmp[j]) j--;
                while(j > i && tmp[i]) i++;
                if(j > i){
                    tmp[j--] = 0;
                    tmp[i++] = 1;
                }
            }
            return tmp.to_ulong();
        }
    };
```

面试题05.06 整数转换（简单）

1. 题目描述

   整数转换。编写一个函数，确定需要改变几个位才能将整数A转成整数B。

```
示例1:
 输入：A = 29 （或者0b11101），B = 15（或者0b01111）
 输出：2

示例2:
 输入：A = 1，B = 2
 输出：2
```

   提示:A，B范围在[-2147483648, 2147483647]之间

2. 简单实现

   注意不要用上题的数1的函数，那个只适用于正数，以后类似的都用bitset比较保险

```
class Solution {
public:
    int convertInteger(int A, int B) {
        int diff = A ^ B;
        bitset<32> d(diff);
        int ans = 0;
        for(int i = 0; i < 32; i++)
            if(d[i] == 1)
                ans++;
        return ans;
    }
};

//自己数的正确方法
class Solution {
    public int convertInteger(int A, int B) {
```

```
        int count = 0;
        for (int n = A ^ B; n != 0; n &= n - 1)//n = n & (n-1)将n的最后一个1变成0
            count++;
        return count;
    }
}
```

面试题05.07 配对交换（简单）

1. 题目描述

   配对交换。编写程序，交换某个整数的奇数位和偶数位，尽量使用较少的指令（也就是说，位0与位1交换，位2与位3交换，以此类推）。

```
示例1:
 输入：num = 2（或者0b10）
 输出 1 （或者 0b01）

示例2:
 输入：num = 3
 输出：3
```

   提示:num的范围在[0, 2^30 - 1]之间，不会发生整数溢出。

2. 简单实现

```
class Solution {
public:
    int exchangeBits(int num) {
        int ans = 0;
        int cnt = 0;//记录已处理对数
        while(num){
            int cur = num & 3;//取得num的最低两位
            if(cur == 1)//01，交换为10
                cur = 2;
            else if(cur == 2)//10，交换为01
                cur = 1;
            ans |= cur << cnt*2;
            cnt++;
            num = num >> 2;
        }
        return ans;
    }
};
```

3. 最优解法

```
class Solution {
    public:
        int exchangeBits(int num) {
            int odd = num & 0x55555555;//取出奇数位，偶数位置0
            int even = num & 0xaaaaaaaa;//取出偶数位，奇数位置0
            odd = odd << 1;//奇数位换到偶数位
            even = even >> 1;//偶数位换到奇数位
            return odd | even;//或运算进行合并
        }
};
```

面试题05.08 绘制直线（中等）

1. t题目描述

   绘制直线。有个单色屏幕存储在一个一维数组中，使得32个连续像素可以存放在一个 int 里。屏幕宽度为w，且w可被32整除（即一个 int 不会分布在两行上），屏幕高度可由数组长度及屏幕宽度推算得出。请实现一个函数，绘制从点(x1, y)到点(x2, y)的水平线。

   给出数组的长度 length，宽度 w（以比特为单位）、直线开始位置 x1（比特为单位）、直线结束位置 x2（比特为单位）、直线所在行数 y。返回绘制过后的数组。

```
示例1:
 输入: length = 1, w = 32, x1 = 30, x2 = 31, y = 0
 输出: [3]
 说明: 在第0行的第30位到第31为画一条直线，屏幕表示为[0b00000000000000000000000000000011]

示例2:
 输入: length = 3, w = 96, x1 = 0, x2 = 95, y = 0
 输出: [-1, -1, -1]
```

2. 简单实现

```
class Solution {
public:
    vector<int> drawLine(int length, int w, int x1, int x2, int y) {
        int cols = w/32;//列数
        vector<int> ans(length, 0);
        int start = y*cols + x1/32;//第一个1所在数在ans中的idx
        int end = y*cols + x2/32;//最后一个1所在数在ans中的idx
        if(start == end){//不跨数
            int cur = 1;
            while(x1 < x2){//构造足够个数的1
                cur = (cur<<1) | 1;
                x1++;
            }
            while(x2 < 31){//补齐右边差的0
                cur = cur<<1;
                x2++;
            }
            ans[start] = cur;
```

```
        }
        else{//跨数
            //第一个数
            int head = 1;
            for(int i = 31 - x1%32; i > 0; i--)
                head = (head<<1) | 1;
            ans[start] = head;
            //中间的数
            for(int i = start+1; i < end; i++)
                ans[i] = -1;
            //最后一个数
            int body = 0x80000000;
            for(int i = x2%32; i > 0; i--)
                body = body >> 1;
            ans[end] = body;
        }

        return ans;
    }
};
```