Table of Contents

1. 编程注意事项

- 思考清楚再开始编码；
- 良好的代码命名和缩进对齐习惯；
- 能够进行单元测试。

2. 编程考察的能力

- 基础知识扎实全面，包括编程语言，数据结构，算法等
- 能写出正确的，完整的，鲁棒的高质量代码
    - 在写代码之前想好测试用例
    - 写完代码之后验证测试用例
- 能思路清晰地分析，解决复杂问题
    - 画图使抽象问题形象化
    - 举例使抽象问题具体化
    - 分析使复杂问题简单化
- 能从时间，空间复杂度两方面优化算法效率
- 具备优秀的沟通能力，学习能力，发散思维能力等

链表

GitBook allows you to organize your book into chapters, each chapter is stored in a separate file like this one.

数组是最简单的一种数据结构，它占据一块连续内存并按照顺序存储数据。由于存储是连续的，可以在O(1)的时间里读/写任何元素，因此效率是非常高的。数组的这个特征非常适合做哈希表，数组的下标最为key，数组存储的内容作为value。

由于数组的O(1)的读写特征，所以有序数组的二分查找和针对下标操作的排序算法也是数组内容的高频考点之一。

11. Container With Most Water

直达：https://leetcode.com/problems/container-with-most-water/description/

Givennnon-negative integers $a_1, a_2, ..., a_n$where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of lineiis at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.

分析

从数组两端开始遍历，每次更改高度最小的那个下标。

C++代码

```
class Solution {
public:
    int maxArea(vector<int>& height) {
        if (height.size() < 2) return 0;
        int max_area = 0;
        int left = 0;
        int right = height.size() - 1;
        while(left < right){
            int area = 0;
            if (height[left] < height[right]){
                area = (right - left) * height[left];
                left++;
            }else{
                area = (right - left) * height[right];
                right --;
            }
            max_area = max_area > area ? max_area:area;
        }
        return max_area;
    }
};
```

16. 3Sum Closet

直达：https://leetcode.com/problems/3sum-closest/description/

Given an arraySofnintegers, find three integers inSsuch that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
    For example, given array S = {-1 2 1 -4}, and target = 1.

    The sum that is closest to the target is 2. (-1 + 2 + 1 = 2)
.
```

分析

问题的核心是两个指针的操作。循环遍历数组，将遍历到的元素取做中心点，从该点的两侧出发初始化left，和right两个指针，如果三个位置只和小于target，则右指针向右移动一位，否则左指针向左移动一位，每次移动如果找到更接近的三组数，则更新返回结果。重复上面过程直到不能移动为止。

C++代码

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int res = nums[0] + nums[1] + nums[2];
        for(int cen = 0; cen < nums.size(); cen++){
            int left = cen - 1;
            int right = cen + 1;
            int sum = nums[left] + nums[cen] + nums[right];
            while( (left >= 0) && (right <= nums.size()-1)){
                sum = nums[left] + nums[cen] + nums[right];
                if(sum > target){
                    if(abs(target-sum)<abs(res-target)) res = su
m;
                    if(left > 0)    left--;
                    else    break;
                }
                else if(sum < target){
                    if(abs(target-sum)<abs(res-target)) res = su
m;
                    if (right < nums.size()-1)   right++;
                    else    break;
                }
                else return target;
            }
        }
        return res;
    }
};
```

118. Pascal's Triangle

直达：https://leetcode.com/problems/pascals-triangle/description/

GivennumRows, generate the firstnumRowsof Pascal's triangle.

For example, givennumRows= 5,
Return

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

分析

帕斯卡三角的特征是

从1开始，以下第i行的第j个元素满足：

1. 如果 j==0 || i == j, pas[i][j] = 1；
2. 否则 pas[i][j] = pas[i-1][j-1] + pas[i][j]

C++代码

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> res;
        for(int i = 0; i<numRows; i++){
            if(i == 0) res.push_back(vector<int>(1,1));
            else{
                vector<int> t;
                for(int j = 0; j<=i; j++){
                    if(j == 0 || j == i)    t.push_back(1);
                    else t.push_back(res[i-1][j-1] + res[i-1][j]
);
                }
                res.push_back(t);
            }
        }
        return res;
    }
};
```

119. Pascal's Triangle II

直达：https://leetcode.com/problems/pascals-triangle-ii/description/

Given an indexk, return thekthrow of the Pascal's triangle.

For example, given k= 3,
Return `[1,3,3,1]` .

Note:
Could you optimize your algorithm to use onlyO(k) extra space?

分析：

类似Q118, 不同之处是可以将中间结果保存在输出向量中。

注：输出向量的第i个元素可以表示为k(k-1)...(k-i)/1*2*...*i, 但这种方法计算可能发生数组越界现象。

C++代码

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        // if(rowIndex == 0) return vector<int>(1,1);
        vector<int> res(rowIndex+1,0);
        for(int i=0;i<rowIndex+2;i++){
            if(i == 0) res[0] = 1;
            else{
                for(int j = i; j>=0; j--){
                    if(j == i || j == 0) res[j] == 1;
                    else res[j] = res[j]+res[j-1];
                }
            }
        }
        return res;
    }
};
```

120. Triangle

暴力递归改动态规划:
<<自底向上搭积木>>

直达：https://leetcode.com/problems/triangle/description/

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

The minimum path sum from top to bottom is `11` (i.e.,2+3+5+1= 11).

Note:
Bonus point if you are able to do this using onlyO(n) extra space, wherenis the total number of rows in the triangle.

分析

算法思路类似于Q119, 不同之处在于矩阵的更新方法，由求和变成求最小值。

C++代码

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        if(triangle.size() == 0) return 0;
        vector<int> res(triangle.size(), 0);
        for(int i = 0; i<triangle.size(); i++){
            if(i == 0) res[0] = triangle[0][0];
            else{
                for(int j=i;j>=0;j--){
                    if(j == i) res[j] = triangle[i][j] + res[j-1
];
                    else if(j == 0) res[j] = triangle[i][j] + re
s[j];
                    else res[j] = min(res[j], res[j-1]) + triang
le[i][j];
                }
            }
        }
        int min_res = res[0];
        for(int k = 0; k<res.size(); k++){
            min_res = min(res[k], min_res);
        }
        return min_res;
    }
};
```

134. Gas Station

直达：https://leetcode.com/problems/gas-station/description/

There areNgas stations along a circular route, where the amount of gas at stationiis `gas[i]` .

You have a car with an unlimited gas tank and it costs `cost[i]` of gas to travel from stationito its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Note:
The solution is guaranteed to be unique.

分析

难点是理清思路，分析出该问题存在的几个性质。

性质1：如果gas[i] < cost[i]，则i不能作为起点；

性质2：对所有路径求和，如果$\sum(gas[i]-cost[i])>0$，则一定存在一点可以遍历整个环。

问题是当确定能够便利之后，如何确定起点：

性质3：如果i能够作为起点，则从i出发可以到任何一个点；

性质4：如果i是唯一的一个点，选择不存在点j可以到i；

性质5：如果i是唯一的一个点，则i一定可以到i+1到n，0到i-1一定不能到i

根据上面三个性质，得到解决问题的思路：

1. 根据性质2，遍历两个数组，确定能不能完成环的遍历；能的话，进入2，否则直接返回-1；
2. 根据性质5，从0出发，查找0不能到的i，如果i+1能到n，则可以返回i+1, 如果i+1不能到n，则将i更新为i能到的最近的节点。

C++代码

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost)
{
        int sum = 0;
        int cur_start = 0;
        int res = 0;
        for (int i=0; i<gas.size(); i++){
            sum += (gas[i] - cost[i]);
            cur_start += (gas[i] - cost[i]);
            if (cur_start < 0){
                cur_start = 0; //重新开始计数
                res = i+1;
            }
        }
        if (sum < 0) return -1;
        else{
            if(res >= gas.size()) return 0;
            else return res;
        }
    }
};
```

2. Add Two Numbers

直达：https://leetcode.com/problems/add-two-numbers/description/

You are given two**non-empty**linked lists representing two non-negative integers. The digits are stored in**reverse order**and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example

```
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
Explanation: 342 + 465 = 807.
```

分析

两链表求和，重点是保存进位标识（carry），每遍历一次就新建一个节点并将新节点插入到结果链表中。

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        int flag = 0; //表示进位
        ListNode res(0);                    这种写法很好
        ListNode *p = &res;
        ListNode *p_head = p;
        while(l1 || l2 || flag){
            int sum = (l1?l1->val:0) + (l2?l2->val:0) + flag;
            flag = sum/10;
            p->next = new ListNode(sum%10);
            p = p->next;
            l1 = l1?l1->next:l1;
            l2 = l2?l2->next:l2;
        }
        return p_head->next;
    }
};
```

19. Remove Nth Node From End of List

直达：https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/

Given a linked list, remove thenthnode from the end of list and return its head.

For example,

```
Given linked list: 1->2->3->4->5, and n = 2.
After removing the second node from the end, the linked list bec
omes
1->2->3->5.
```

Note:
Given n will always be valid. Try to do this in one pass.

分析

两个指针p, end，end走了n步之后p再开始走，另外还需要一个prior指针跟在p的前面，以便用处链表节点的删除。

重点在于边界情况的处理，即n等于链表的长度。

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* p = head;
        ListNode* end = head;
        ListNode* prior = head;
        int cnt = 0;
        while(end -> next){
            if(cnt >= n-1){
                end = end -> next;
                prior = p;
                p = p -> next;
                cnt++;
            }else{
                end = end -> next;
            }
            cnt++;
        }
        if(p == prior){
            head = head -> next;
        }else{
            prior -> next = p ->next;
        }
        return head;
    }
};
```

82. Remove Duplicates from Sorted List II

直达：https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving onlydistinctnumbers from the original list.

For example,
Given `1->2->3->3->4->4->5` , return `1->2->5` .
Given `1->1->1->2->3` , return `2->3` .

分析

增加两个辅助指针，d_head作为伪头节点，p_haed辅助删除数据

需要注意的几个边界情况及解决方案

1. 头节点需要删除：设置伪头节点，伪头节点的下个元素是head，返回的时候返回d_head的下一个节点
2. 最后一个节点需要删除：判断是否是最后一个节点，如果是的话将p指向空，返回结果

C++代码

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(head == NULL) return head;
        ListNode* d_head = new ListNode(0);
        ListNode* p_head = d_head;
        d_head -> next = head;
        while(head->next){
            if(head->val == head->next->val){
                int del_val = head->val;
                while(head->val == del_val){
                    if(head->next) head = head->next;
                    else{
                        p_head->next = NULL;
                        return d_head->next;
                    }
                }
                p_head->next = head;
            }else{
                p_head = head;
                head = head->next;
            }
        }
        return d_head->next;
    }
};
```

86. Partition List

直达: https://leetcode.com/problems/partition-list/description/

Given a linked list and a valuex, partition it such that all nodes less thanxcome before nodes greater than or equal tox.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given `1->4->3->2->5->2` andx= 3,
return `1->2->2->4->3->5` .

给定一个整数链表和一个整数x，将小于x的移到新链表的左边，大于等于的移到右边，保证链表的同一部分的顺序不变，如上图。

分析

初始化两个链表，一个保存小于x的，一个保存大于等于x的，最后重新连接两个链表即可。

算法

```
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode* l1 = new ListNode(0);
        ListNode* l2 = new ListNode(0);
        ListNode* l1_2 = l1;
        ListNode* l2_2 = l2;
        while(head){
            if(head->val >= x){
                l2_2 -> next = head;
                l2_2 = l2_2->next;
            }else{
                l1_2 ->next = head;
                l1_2 = l1_2->next;
            }
            head = head->next;
        }
        l2_2->next = NULL;
        l1_2->next = l2->next;
        return l1->next;
    }
};
```

92. Reverse Linked List II

直达：https://leetcode.com/problems/reverse-linked-list-ii/description/

Reverse a linked list from positionmton. Do it in-place and in one-pass.

For example:
Given `1->2->3->4->5->NULL` , m = 2 and n = 4,

return `1->4->3->2->5->NULL` .

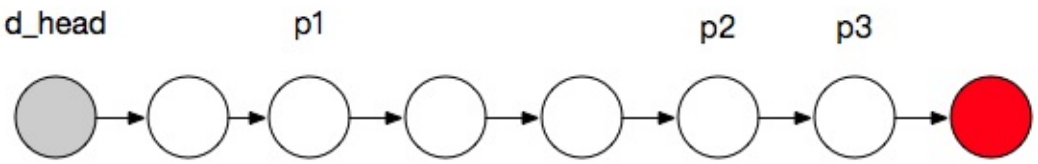
Note:
Givenm,nsatisfy the following condition:
1 ≤m≤n≤ length of list.

分析

伪头节点在链表题目中很有用，
可以针对大多数的特殊情况处理

题目要求一次遍历，且不能使用额外的存储空间。首先，设置一个伪头节点dhead（防止m=1）以及三个指针p1 = p2 =d_head, p3 = head。开始从i=1开始遍历到n:

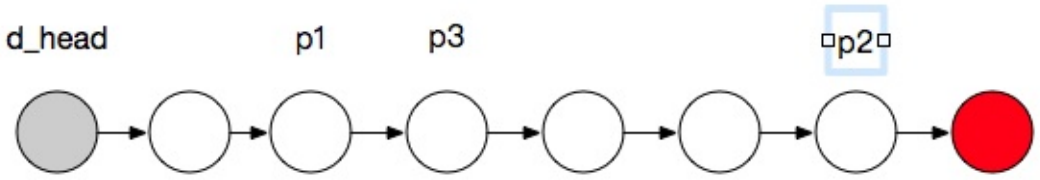
1. 当i<m时，三个指针都向前移动一位
2. 当i=m是，p1停止遍历，此时p1后面就是要反转的链表，p2，p3继续遍历
3. 当i>m且i<=n时，此时需要交换节点，即把p3指向的节点插入到p1后面，如下图



接下来就是链表节点的移动了

```
p2->next = p3->next;
p3->next = p1->next;
p1->next = p3;
```

移动后如下图



由于不知道红色节点是否存在，所以需要判断p2->next用于决定p3的值

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        ListNode* d_head = new ListNode(0);
        d_head -> next = head;
        ListNode* p1 = d_head;
        ListNode* p2 = d_head;
        ListNode* p3 = head;
        for(int i = 1; i<=n; i++){
            if(i < m){
                p1 = p1->next;
                p2 = p2->next;
                p3 = p3->next;
            }else if (i == m){
                p2 = p2->next;
                p3 = p3->next;
            }else{
                p2->next = p3->next;
                p3->next = p1->next;
                p1->next = p3;
                if(p2->next){
                    p3 = p2->next;
                }
            }
        }
        return d_head -> next;

    }
};
```

1. 双指针不仅能用在数组上，
也可以用在链表问题上；
2. 双指针不仅有方向性，还有速度

141. Linked List Cycle

直达：https://leetcode.com/problems/linked-list-cycle/description/

Given a linked list, determine if it has a cycle in it.

Follow up:
Can you solve it without using extra space?

分析

设置两个指针，快指针每次遍历两个节点，慢指针每次遍历一个节点，如果快指针能和慢指针相遇，则说明存在环。注意单独处理空链表的情况。

C++代码

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(!head) return false;
        ListNode* slow = head;
        ListNode* fast = head;
        while(fast && fast->next){
            fast = fast->next->next;
            slow = slow->next;
            if(fast == slow) return true;
        }
        return false;
    }
};
```

142. Linked List Cycle II

直达：https://leetcode.com/problems/linked-list-cycle-ii/description/

Given a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

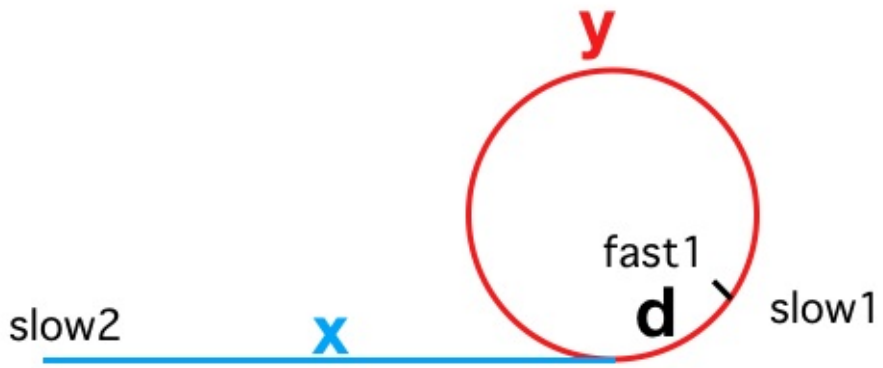
Note: Do not modify the linked list.

Follow up:
Can you solve it without using extra space?

分析

设置三个指针，快指针fast每次遍历两个节点，慢指针slow1，slow2每次遍历两个节点。

开始的时候fast和slow1先从头节点开始遍历，相遇的时候fast走的距离是slow1的两倍，如下图:



假设非环链表部分的长度是x，环的周长是y，fast和slow1在d处相遇，此时fast比slow1多走了一个圆环，于是有

2 * (x+d) = x+d+y

化简得：

x = y-d

第二轮遍历的时候slow2从头结点开始遍历，slow1从d处开始遍历，当遍历到环起始点的时候，因为x=y-d，所以两指针会在这个地方相遇。

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        if(!head) return NULL;
        ListNode* slow1 = head;
        ListNode* slow2 = head;
        ListNode* fast = head;
        while(fast && fast->next){
            fast = fast->next->next;
            slow1 = slow1->next;
            if(fast == slow1){
                while(slow2 != slow1){
                    slow2 = slow2->next;
                    slow1 = slow1->next;
                }
                return slow2;
            }
        }
        return NULL;
    }
};
```

147. Insertion Sort List

直达：https://leetcode.com/problems/insertion-sort-list/description/

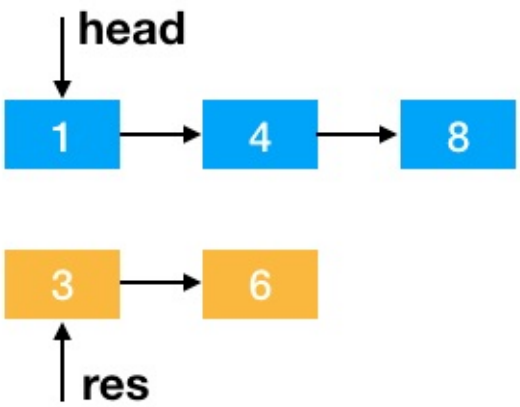
Sort a linked list using insertion sort.

分析

知识点在于链表的插入，分成两种情况。

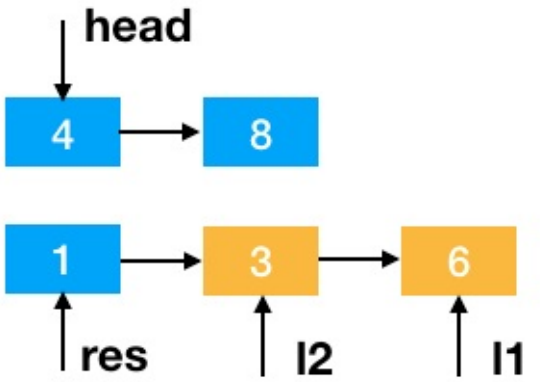
1. 在头结点之前插入

如下图在3之前插入1：




```
ListNode* tmp = head;
head = head->next;
tmp->next = res;
res = tmp;
```

1. 在中间或者尾部插入，如下图



首先顺序遍历res链表，找到该插入的位置，即将head插入l2和l1之间

```
l2->next = tmp;
head = head->next;
tmp->next = l1;
```

C++代码

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {
        if(head==NULL || head->next==NULL) return head;
        ListNode* res = head;
        head = head->next;
        res->next = NULL;
        while(head){
            ListNode* tmp = head;
            if(head->val <= res->val){
                head = head->next;
                tmp->next = res;
                res = tmp;
            }else{
                ListNode* l1 = res;
                ListNode* l2 = l1;
                while(l1 && l1->val < head->val){
                    l2 = l1;
                    l1 = l1->next;
                }
                l2->next = tmp;
                head = head->next;
                tmp->next = l1;
            }
        }
        return res;
    }
};
```

160. Intersection of Two Linked Lists

直达：https://leetcode.com/problems/intersection-of-two-linked-lists/description/

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

```
A:          a1 → a2
                   ↘
                     c1 → c2 → c3
                   ↗
B:     b1 → b2 → b3
```

begin to intersect at node c1.

Notes:

- If the two linked lists have no intersection at all, return `null`
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

分析

有限次数遍历的复杂度仍旧是O(n).

1. 遍历两个数组的长度;
2. 长数组比短数组长的部分一定不可能是交点，可先遍历长数组，知道剩下的长度等于短数组；
3. 同事遍历两个数组并比较是否是交点。

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *getIntersectionNode(ListNode *headA, ListNode *hea
dB) {
        if (headA==NULL || headB==NULL) return NULL;
        int lenA = 0, lenB = 0;
        ListNode* lA = headA;
        ListNode* lB = headB;
        while(lA){
            lA = lA->next;
            lenA++;
        }
        while(lB){
            lB = lB->next;
            lenB++;
        }
        lA = headA; lB = headB;
        if(lenA > lenB){
            for(int i = 0; i<lenA-lenB; i++) lA = lA->next;
        }else{
            for(int i = 0; i<lenB-lenA; i++) lB = lB->next;
        }
        while(lA != lB){
            lA = lA->next;
            lB = lB->next;
        }
        return lA;
    }
};
```

206. Reverse Linked List

直达：https://leetcode.com/problems/reverse-linked-list/description/

Reverse a singly linked list

分析

分成两种情况

1. 如果链表的节点数小于等于1个，直接返回链表即可
2. 如果大于1个，首先初始化几个指针用于遍历，如下图

```
head  p
 |    |
 1 -> 2 -> 3 -> 4 -> 5
 |
 l
```

然后判断什么时候到达链表尾部

1.1 如果没有到达尾部

```
l = p->next;
p->next = head;
head = p;
p = l;
```

1.2 如果到了尾部

```
p->next = head;
head = p;
return head;
```

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if(head == NULL || head->next==NULL) return head;
        ListNode* l = NULL;
        ListNode* p = head->next;
        head->next = NULL;
        while(p){
            if(p->next!=NULL){
                l = p->next;
                p->next = head;
                head = p;
                p = l;
            }else{
                p->next = head;
                head = p;
                return head;
            }
        }
    }
};
```

1. Two Sum

直达：https://leetcode.com/problems/two-sum/description/

Given an array of integers, return**indices**of the two numbers such that they add up to a specific target.

You may assume that each input would have**exactly**one solution, and you may not use thesameelement twice.

Example:

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

分析

用哈希表存储已经遍历的元素(hash[key]=val), 如果遇到和满足target的，则将两个数返回，否则返回空

C++代码

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map <int, int> hash;
        vector<int> res;
        for(int i=0; i<nums.size(); i++){
            if (hash.find(target - nums[i]) != hash.end()){
                res.push_back(hash[target-nums[i]]);
                res.push_back(i);
                return res;
            }else{
                hash[nums[i]] = i;
            }
        }
        return res;
    }
};
```

3. Longest Substring Without Repeating Characters

直达：https://leetcode.com/problems/longest-substring-without-repeating-characters/description/

Given a string, find the length of the**longest substring**without repeating characters.

Examples:

Given `"abcabcbb"` , the answer is `"abc"` , which the length is 3.

Given `"bbbbb"` , the answer is `"b"` , with the length of 1.

Given `"pwwkew"` , the answer is `"wke"` , with the length of 3. Note that the answer must be a**substring**, `"pwke"` is asubsequenceand not a substring.

分析

用哈希表存储每个元素的下标（ `hash[s[i]] = i` ）如重复，可用新的下标覆盖旧的下标。遍历字符串，如果哈希表中不存在该字符，

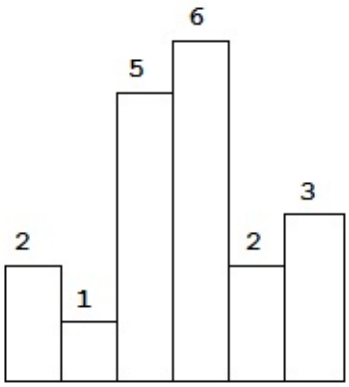
C++代码

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        unordered_map <char, int> hash;
        int res = 0;
        int cur_res = 0;
        for (int i = 0; i<s.size(); i++){
            if (hash.find(s[i]) == hash.end()){
                cur_res += 1;
            }else{
                cur_res = min((i-hash[s[i]]), (cur_res+1));
            }
            hash[s[i]] = i;
            res = res>cur_res?res:cur_res;
        }
        return res;
    }
};
```

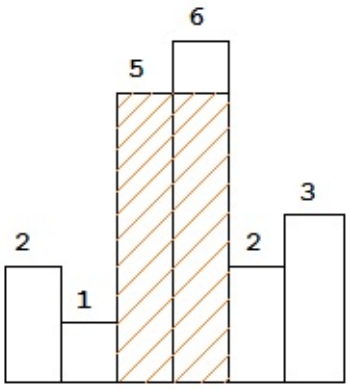
84. Largest Rectangle in Histogram

直达：https://leetcode.com/problems/largest-rectangle-in-histogram/description/

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.



Above is a histogram where width of each bar is 1, given height = `[2,1,5,6,2,3]`.



The largest rectangle is shown in the shaded area, which has area = `10` unit.

For example,
Given heights = `[2,1,5,6,2,3]`,
return `10`.

分析：

1. 计算以每个bar为高的最大矩形区域面积，返回最大值；
2. 如果bar的高度逐渐增高，则围城的面积逐渐增大，不需要计算；

3. 如果bar的高度开始递减（设为k），则要计算递减之前的最大面积，为了避免重复计算，从右向左计算至小于height[k]的时候停止计算即可；
4. 可在数组后面添加一个高度为0的bar，用于辅助清空临时堆栈；
5. 从左向右遍历，再从右向左计算面积，明显用堆栈存储效率更高，由于可以通过下标访问到高度，存储下标即可。

举例：

1. 索引i=0, height[0] = 2，堆栈为空，直接将2的下标0压栈，i++；
2. i=1，height[1]=1 < 2, 开始从右向左计算面积；
    i. i=0的bar的面积是0的长度1乘以高度2，即为2，保存res=2；
    ii. 堆栈为空，停止弹出。
3. i=1, 堆栈为空，将i=1压栈, i++；
4. i=2, height[i]=5>height[stk.top()]=1, 将i=2压栈，i++;
5. i=3, height[i]=6>height[stk.top()]=5. 将i=3压栈，i++, 此时，stk=[1,2,3];
6. i=4, 此时height[i]=2<height[stk.top()], 开始从右向左计算面积；
    i. 弹出栈顶h=stk.top()=3, stk.pop(), 此时的矩形区域的长是(i-stk.top()-1), 高是height[h], area = (i-stk.top()-1)*height[h] = 6, res更新为6;
    ii. 弹出栈顶h=stk.top()=2, area = (i-stk.top()-1) * height[h] = (4-1-1)*5 = 10.
7. i=4, stk.top()=1 < height[i]=2, 4入栈，i++;
8. i=5, 5入栈，i++, 此时stk=[1,4,5];

i=6, 遍历到添加的0高度，开始从右向左计算面积，并清空堆栈；

1. h=stk.top()=5, stk.pop(), area = (6-stk.top()-1) * height[5] = 3, stk = [1,4], res = 10;
2. h=stk.top()=4, stk.pop(), area = (6-stk.top()-1) * height[4] = 8, stk = [1], res = 10;
3. h=stk.top()=1, stk.pop(), 堆栈为空，area = (i)*height[h] = 6, res = 10;
4. 由此可以返回最大值10

C++实现：

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        stack<int> stk;
        heights.push_back(0);
        int res = 0;
        for (int i = 0; i < heights.size(); i++){
            if(stk.empty() || heights[i] > heights[stk.top()]){
                stk.push(i);
            }else{
                int h = stk.top();
                stk.pop();
                res = max(res, (stk.empty()?i:i-stk.top()-1) * h
eights[h]);
                i--;
            }
        }
        return res;
    }
};
```

155. Min Stack

直达：https://leetcode.com/problems/min-stack/description/

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

Example:

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

分析

栈的压入和弹出过程可能会导致堆栈最小值发生变化，因此需要一个结构来保存当前堆栈的最小值。

1. 例如数组$[3, 2, 4, 5, ..., a_n]$（其中省略号的数都大于2），此时无虑在最小值1后面的数是怎样变化的，getMin()返回的最小值都是2。
2. 但是如果省略号中存在一个1，例如[3, 2, 4, 5, ..., 1, ..., a_n], 此时第二个省略号无论怎么变化，返回的都是1，第一个省略号内的内容无论怎么变化，返回的都是2.
3. 同理，如果第二个省略号中含有更小的值，情况将类似2。

所以，根据分析：

1. 根据3，数据结构应该采用堆栈保存最小值
2. 根据2，如果出栈的值等于最小值，则要更新最小堆栈，即弹出栈顶元素。
3. 根据1，如果入栈的值小于等于当前最小值，则最小堆栈压入该元素。
    i. 为什么要有等于呢？考虑下面情况，例如依次将数组[3, 2, 4, 5, 7, 2]入栈，如果没有等于的话，最小栈中保存的元素应该是[3, 2], 堆栈的元素是[3,2,4,5,7,2], 堆栈弹出一个元素，由于弹出的元素等于getMin(), 根据2，最小栈也要弹出栈顶，此时最小栈为[3], 显然是不正确的。

C++代码

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

1. Valid Parentheses

Q255. Implement Stack using Queues

Implement the following operations of a stack using queues.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

Notes:

- You must use only standard operations of a queue -- which means only `push to back`, `peek/pop from front`, `size`, and `is empty` operations are valid.
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
- You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

分析

用一个队列实现即可，stack和queue的区别在于弹出的顺序不同。所以在每次插入时，通过循环的方式将新的元素移动到最后的位置即可。

C++代码

```
class MyStack {
public:
    /** Initialize your data structure here. */
    queue<int> que1;
    MyStack() {

    }
```

```
    /** Push element x onto stack. */
    void push(int x) {
        que1.push(x);
        for(int i=0; i< que1.size()-1; i++){
            que1.push(que1.front());
            que1.pop();
        }
    }

    /** Removes the element on top of the stack and returns that
 element. */
    int pop() {
        int res = que1.front();
        que1.pop();
        return res;
    }

    /** Get the top element. */
    int top() {
        return que1.front();
    }

    /** Returns whether the stack is empty. */
    bool empty() {
        return que1.empty();
    }
};

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStack obj = new MyStack();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.top();
 * bool param_4 = obj.empty();
 */
```

Q232. Implement Queue using Stacks

直达: https://leetcode.com/problems/implement-queue-using-stacks/description/

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

Notes:

- You must use only standard operations of a stack -- which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

分析：

堆栈的操作顺序是先进后出，而队列是先进先出，如果要用堆栈实现队列，要在push, pop, top和empty四个操作中满足满足队列的先进先出的顺序

pop

举个例子，队列中先后插入了3个元素a,b,c, 首先我们把它们都存到栈stk1中，此时若要弹出一个数，按照堆栈的先进后出的顺序，应该弹出c，而队列的弹出应该是a。这时候，堆栈stk2就可以起作用了，我们将stk1的所有元素全部移动到stk2中，这时候再从stk2中弹出栈顶元素即可。

```
stk1:
stk2: c,b
```

而插入的时候我们只需要将要插入的元素直接插入到stk1，看看能否只在push操作中移动堆栈元素。继续上面的例子，此时来了一个新的元素d，插入到stk1中。当要弹出元素时，由于剩下的元素中b是先来的，所以直接弹出b的栈顶元素即可，不需要移动stk1的元素，只有当stk2中为空时，才需要将stk1的元素移动到stk2中。

```
stk1: d
stk2: c,b
```

总结：

如果stk2为空，则将stk1的所有元素移动到stk2中，并弹出stk2的栈顶元素

如果stk2不为空，则直接弹出stk2的栈顶元素

push

插入到stk1中，不需要任何其他操作

top

类似pop操作，只是不弹出stk2的栈顶元素，只需要返回栈顶值即可

empty

stk1和stk2都为空，则队列为空

C++代码

```
class MyQueue {
    stack<int> stk1;
    stack<int> stk2;
public:
    /** Initialize your data structure here. */
    MyQueue() {
    }

    /** Push element x to the back of queue. */
    void push(int x) {
        stk1.push(x);
    }
```

```
    /** Removes the element from in front of queue and returns t
hat element. */
    int pop() {
        int res = peek();
        stk2.pop();
        return res;
    }

    /** Get the front element. */
    int peek() {
       if(!stk2.empty()){
            return stk2.top();
        }else{
            while(!stk1.empty()){
                stk2.push(stk1.top());
                stk1.pop();
            }
            return stk2.top();
        }
    }

    /** Returns whether the queue is empty. */
    bool empty() {
        return stk1.empty() && stk2.empty();
    }
};

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue obj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * bool param_4 = obj.empty();
 */
```

94. Binary Tree Inorder Traversal

直达：https://leetcode.com/problems/binary-tree-inorder-traversal/description/

Given a binary tree, return theinordertraversal of its nodes' values.

For example:
Given binary tree `[1,null,2,3]` ,

```
   1
    \
     2
    /
   3
```

return `[1,3,2]` .

Note: Recursive solution is trivial, could you do it iteratively?

分析

树的中序遍历是先遍历左子树，输出根节点后再遍历右节点，递归的进行即可。

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        if(root) helper(res, root);
        return res;
    }
private:
    void helper(vector<int>& res, TreeNode* root){
        if(root->left) helper(res, root->left);
        res.push_back(root->val);
        if(root->right) helper(res, root->right);
    }
};
```

100. Same Tree

直达：https://leetcode.com/problems/same-tree/description/

Given two binary trees, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

Example 1:

```
Input:     1         1
          / \       / \
         2   3     2   3

        [1,2,3],   [1,2,3]

Output: true
```

Example 2:

```
Input:     1         1
          /           \
         2             2

        [1,2],     [1,null,2]

Output: false
```

Example 3:

```
Input:     1         1
          / \       / \
         2   1     1   2

        [1,2,1],   [1,1,2]

Output: false
```

分析

两棵树相等的充要条件是根节点内容相同且左右两棵字数都是相同的树

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p==NULL && q==NULL) return true;
        if (p==NULL ^ q==NULL) return false;
        if (p->val != q->val) return false;
        else return isSameTree(p->left, q->left) && isSameTree(p
->right, q->right);
    }
};
```

101. Symmetric Tree

直达：https://leetcode.com/problems/symmetric-tree/description/

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree `[1,2,2,3,4,4,3]` is symmetric:

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

But the following `[1,2,2,null,3,null,3]` is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

Note:
Bonus points if you could solve it both recursively and iteratively.

分析

两棵树是对称的充要条件是第一棵树的左子树和第二棵树的右子树互相对称，并且第一棵树的右子树与第二棵树的左子树互相对称。

所以用一个判断两个数是否对称做辅助函数

C++代码

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
        if(root==NULL) return true;
        return symmetricTrees(root->left, root->right);
    }
private:
    bool symmetricTrees(TreeNode* left, TreeNode* right){
        if(left==NULL && right==NULL) return true;
        if(left==NULL ^ right==NULL) return false;
        if(left->val != right->val) return false;
        if(!symmetricTrees(left->left, right->right)) return fal
se;
        if(!symmetricTrees(left->right, right->left)) return fal
se;
        return true;
    }
};
```

102. Binary Tree Level Order Traversal

直达：https://leetcode.com/problems/binary-tree-level-order-traversal/description/

Given a binary tree, return thelevel ordertraversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree `[3,9,20,null,null,15,7]` ,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

分析

添加一个变量，用于保存当前遍历的层数，根据层数在输出矩阵中将遍历到的节点值添加进去。

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if(root==NULL) return res;
        helper(res, root, 0);
        return res;
    }
private:
    void helper(vector<vector<int>>& res, TreeNode* root, int po
s){
        if(res.size() <= pos){
            res.push_back(vector<int>(0,0));
        }
        res[pos].push_back(root->val);
        if(root->left) helper(res, root->left, pos+1);
        if(root->right) helper(res, root->right, pos+1);
    }
};
```

93. Binary Tree Zigzag Level Order Traversal

直达：https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/

Given a binary tree, return thezigzag level ordertraversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:
Given binary tree `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

分析

Q102层次遍历的变形，用堆栈重新排列Q102结果的行数摩2等于1的那一行元素。

C++代码

```
/**
 * Definition for a binary tree node.
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> res;
        if(root==NULL) return res;
        helper(res, root, 0);
        for(int i=0; i<res.size(); i++){
            if(i%2 == 1){
                stack<int> stk;
                for(int j=0; j<res[i].size();j++){
                    stk.push(res[i][j]);
                }
                int k = 0;
                while(!stk.empty()){
                    res[i][k] = stk.top();
                    k++;
                    stk.pop();
                }
            }
        }
        return res;
    }
private:
    void helper(vector<vector<int>>& res, TreeNode* root, int po
s){
        if(res.size() <= pos){
            res.push_back(vector<int>(0,0));
        }
        res[pos].push_back(root->val);
        if(root->left) helper(res, root->left, pos+1);
        if(root->right) helper(res, root->right, pos+1);
    }
};
```

104. Maximum Depth of Binary Tree

直达：https://leetcode.com/problems/maximum-depth-of-binary-tree/description/

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

分析：

深度优先遍历树，保存两个变量，一个是当前节点的深度，另外一个是已经遍历的树节点的最大的深度。返回最大深度即可。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        int h = 0;
        int res = 0;
        if(!root) return 0;
        helper(root, h, res);
        return res;
    }
private:
    void helper(TreeNode* root, int h, int& res){
        if(root==NULL) res = max(h, res);
        else{
            helper(root->left, h+1, res);
            helper(root->right, h+1, res);
        }
    }
};
```

105. Construct Binary Tree from Preorder and Inorder Traversal

直达：https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/

Given preorder and inorder traversal of a tree, construct the binary tree.

分析

对于先序遍历和中序遍历的两串字符串，先序遍历的第一个字符便是当前树的根节点。为了迭代的使用该算法，需要确定左右两棵子树的先序遍历和中序遍历序列。

对于中序遍历，从当前字符串中找到先序遍历序列的第一个字符，当前字符左侧便为左子树的中序遍历序列（假设有m个字符），右侧便是右子树的中序遍历序列。

对于先序遍历，从第一个元素开始的m个元素是左子树的先序遍历，剩下的便是右子树的先序遍历。

为了节约存储空间，可以构建全局变量的hash表存储树节点在中序遍历序列中的位置。

C++代码：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    unordered_map<int, int> hash;
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inor
der) {
        if(preorder.empty()) return NULL;
        for(int i = 0; i < inorder.size(); i++) hash[inorder[i]]
 = i;
        return helper(preorder, inorder, 0, preorder.size() - 1,
0, inorder.size() - 1);
    }
private:
    TreeNode* helper(vector<int>& preorder, vector<int>& inorder
, int s0, int e0, int s1, int e1){
        if(s0 > e0 || s1 > e1) return NULL;
        int mid = hash[preorder[s0]];
        TreeNode* root = new TreeNode(preorder[s0]);
        int num = mid-s1;
        root->left = helper(preorder, inorder, s0+1, s0+num, s1,
 mid-1);
        root->right = helper(preorder, inorder, s0+num+1, e0, mi
d+1, e1);
        return root;
    }
};
```

106. Construct Binary Tree from Inorder and Postorder Traversal

直达：https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/

Given inorder and postorder traversal of a tree, construct the binary tree.

分析

和Q105思想相同。

需要注意的是根节点是后序遍历序列的最后一个元素。

左，右子树的中序和Q105取法相同。

左子树的后续遍历是开始到m-1个字符，右子树的后序遍历是m到倒数第二个字符

C++代码：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    unordered_map<int, int> hash;
    TreeNode* buildTree(vector<int>& inorder, vector<int>& posto
rder) {
        if(inorder.empty()) return NULL;
        for(int i = 0; i < inorder.size(); i++) hash[inorder[i]]
 = i;
        return helper(inorder, postorder, 0, inorder.size() - 1,
0, postorder.size() - 1);
    }
private:
    TreeNode* helper(vector<int>& inorder, vector<int>& postorde
r, int s0, int e0, int s1, int e1){
        if(s0 > e0 || s1 > e1) return NULL;
        int mid = hash[postorder[e1]];
        TreeNode* root = new TreeNode(postorder[e1]);
        int num = mid-s0;
        root->left = helper(inorder, postorder, s0, mid-1, s1, s
1+num-1);
        root->right = helper(inorder, postorder, mid+1, e0, s1+n
um, e1-1);
        return root;
    }
};
```

107. Binary Tree Level Order Traversal II

直达：https://leetcode.com/problems/binary-tree-level-order-traversal-ii/description/

Given a binary tree, return thebottom-up level ordertraversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example:
Given binary tree `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]
]
```

分析

和Q102思想一样，只需用堆栈重新将数组重新排序即可。

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {
        vector<vector<int>> res1;
        vector<vector<int>> res2;
        if(root==NULL) return res1;
        helper(res1, root, 0);
        stack<vector<int>> stk;
        for(int i = 0; i<res1.size(); i++){
            stk.push(res1[i]);
        }
        while(!stk.empty()){
            res2.push_back(stk.top());
            stk.pop();
        }
        return res2;
    }
private:
    void helper(vector<vector<int>>& res, TreeNode* root, int po
s){
        if(res.size() <= pos){
            res.push_back(vector<int>(0,0));
        }
        res[pos].push_back(root->val);
        if(root->left) helper(res, root->left, pos+1);
        if(root->right) helper(res, root->right, pos+1);
    }
};
```

108. Convert Sorted Array to Binary Search Tree

直达：https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/description/

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees ofeverynode never differ by more than 1.

Example:

```
Given the sorted array: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents th
e following height balanced BST:

      0
     / \
   -3   9
   /   /
```

分析

在有序数组中，去中间元素作为根节点，左侧序列迭代的构建新的子树作为根节点的左子树，右侧部分作为右子树。递归进行直到序列为空。

C++代码

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return helper(nums, 0, nums.size()-1);
    }
private:
    TreeNode* helper(vector<int>& nums, int left, int right){
        if(left > right) return NULL;
        int mid = (right-left)/2+left;
        TreeNode* root = new TreeNode(nums[mid]);
        root->left = helper(nums, left, mid-1);
        root->right = helper(nums, mid+1, right);
        return root;
    }
};
```

109. Convert Sorted List to Binary Search Tree

直达：https://leetcode.com/problems/convert-sorted-list-to-binary-search-tree/description/

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees ofeverynode never differ by more than 1.

Example:

```
Given the sorted linked list: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents th
e following height balanced BST:

      0
     / \
   -3   9
   /   /
 -10  5
```

分析

和Q108一样，重点是如何确定链表中间元素以及如何切割链表。

用两个指针遍历，快指针fast每次走两步，慢指针slow每次走一步，当快指针走到结尾处（或者倒数第二个）尾指针正好走到中间（或者左偏一格），则slow的下一个便是中间指针，如下图。

```
     slow  fast
      |     |
-10, -3, 0, 5, 9
 |        |
head     mid
```

将mid作为根节点，head到slow作为左子树链表，mid->next到最后作为右子树的链表，递归进行直到链表为空或者只剩一个元素的时候返回即可。

C++代码

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* sortedListToBST(ListNode* head) {
        if (!head) return NULL;
        if (!head->next) return new TreeNode(head->val);
        ListNode* slow = head;
        ListNode* fast = head->next;
        while(fast->next && fast->next->next){
            fast = fast->next->next;
            slow = slow->next;
        }
        ListNode* mid = slow->next;
        TreeNode* root = new TreeNode(mid->val);
        l1->next = NULL;
        root->left = sortedListToBST(head);
        root->right = sortedListToBST(mid->next);
        return root;
    }
};
```

110. Balanced Binary Tree

直达：https://leetcode.com/problems/balanced-binary-tree/description/

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees ofeverynode never differ by more than 1.

分析

平衡二叉树的两棵子树的高度差小于2且左右两棵子树都是平衡二叉树，递归的判断即可。另外，树的高度也可以通过递归进行计算。

C++代码：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isBalanced(TreeNode* root) {
        if(!root) return true;
        if(abs(treeHeight(root->left) - treeHeight(root->right))
 >= 2) return false;
        return isBalanced(root->left) && isBalanced(root->right)
;
    }
private:
    int treeHeight(TreeNode* root){
        if (!root) return 0;
        return max(treeHeight(root->left), treeHeight(root->righ
t))+1;
    }
};
```

111. Minimum Depth of Binary Tree

直达：https://leetcode.com/problems/minimum-depth-of-binary-tree/description/

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

分析

使用中间变量保存所有遍历的叶节点的最小深度，递归的遍历即可。

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int minDepth(TreeNode* root) {
        int res = INT_MAX;
        if(!root) return 0;
        treeHeight(root, 1, res);
        return res;
    }
private:
    void treeHeight(TreeNode* root, int height, int& minHeight){
        if(!root->left && !root->right){
            minHeight = min(height, minHeight);
            return;
        }
        if(root->left) treeHeight(root->left, height+1, minHeigh
t);
        if(root->right) treeHeight(root->right, height+1, minHei
ght);
    }
};
```

112. Path Sum

直达：https://leetcode.com/problems/path-sum/description/

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

For example:

Given the below binary tree and

`sum = 22`

```
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \      \
        7    2      1
```

return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

分析

保存全局变量path计算根节点到该节点的路径长度，递归的计算即可。另外，用全局变量保存叶节点的路径长度是否是sum。

C++代码

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool hasPathSum(TreeNode* root, int sum) {
        if(!root) return false;
        bool res = false;
        treePath(root, root->val, sum, res);
        return res;
    }
private:
    void treePath(TreeNode* root, int path, int sum, bool& res){
        if(!root->left && !root->right){
            res = res || (path == sum);
            return;
        }
        if(root->left) treePath(root->left, path+root->left->val
, sum, res);
        if(root->right) treePath(root->right, path+root->right->
val, sum, res);
    }
};
```

113. Path Sum II

直达：https://leetcode.com/problems/path-sum-ii/description/

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

For example:

Given the below binary tree and

`sum = 22`,

```
              5
             / \
            4   8
           /   / \
          11  13  4
         /  \    / \
        7    2  5   1
```

return

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

分析

类似于Q112, 只需要保存一个变量path记录当前遍历到的节点，迭代的遍历到叶节点之后如果满足路径之和等于sum后将路径插入返回结果中即可。

C++代码

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

114. Flatten Binary Tree to Linked List

直达：https://leetcode.com/problems/flatten-binary-tree-to-linked-list/description/

Given a binary tree, flatten it to a linked list in-place.

For example,
Given

```
          1
         / \
        2   5
       / \   \
      3   4   6
```

The flattened tree should look like:

```
   1
    \
     2
      \
       3
        \
         4
          \
           5
            \
             6
```

分析

如上图，首先将左右子树flatten，然后再转换成程序的输出。同样，左右子树也可以通过这个方法进行flat。如下图所示。所以，该题可以通过递归进行计算。转换方式是将左子树设成根节点的右子树，将右子树接到原左子树的叶节点的地方。

```
  1
 / \
2   5
 \   \
  3   6
   \
    4
```

C++代码

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void flatten(TreeNode* root) {
        if(!root || !root->left && !root->right) return;
        flatten(root->left);
        flatten(root->right);
        TreeNode* temp_right = root->right;
        root->right = root->left;
        root->left = NULL;
        while(root->right) root = root->right;
        root->right = temp_right;
    }
};
```

116. Populating Next Right Pointers in Each Node

直达： https://leetcode.com/problems/populating-next-right-pointers-in-each-node/description/

Given a binary tree

```
struct TreeLinkNode {
  TreeLinkNode *left;
  TreeLinkNode *right;
  TreeLinkNode *next;
}
```

Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to `NULL`.

Initially, all next pointers are set to `NULL`.

Note:

- You may only use constant extra space.
- You may assume that it is a perfect binary tree (ie, all leaves are at the same level, and every parent has two children).

For example,
Given the following perfect binary tree,

```
     1
   /  \
  2    3
 / \  / \
4  5  6  7
```

After calling your function, the tree should look like:

```
        1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \  / \
    4->5->6->7 -> NULL
```

分析

对于完美的二叉树来说，左子树的next为其父节点的右子树，右节点的next是其root->next的左子树，依照此规律，递归的构建即可。

C++代码

```
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 *  int val;
 *  TreeLinkNode *left, *right, *next;
 *  TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(
NULL) {}
 * };
 */
class Solution {
public:
    void connect(TreeLinkNode *root) {
        if(!root) return;
        if(root->left) root->left->next = root->right;
        if(root->right && root->next) root->right->next = root->
next->left;
        connect(root->left);
        connect(root->right);
    }
};
```

117. Populating Next Right Pointers in Each Node II

直达：https://leetcode.com/problems/populating-next-right-pointers-in-each-node-ii/description/

Follow up for problem "Populating Next Right Pointers in Each Node".

What if the given tree could be any binary tree? Would your previous solution still work?

Note:

- You may only use constant extra space.

For example,
Given the following binary tree,

```
         1
       /  \
      2    3
     / \    \
    4   5    7
```

After calling your function, the tree should look like:

```
         1 -> NULL
       /  \
      2 -> 3 -> NULL
     / \    \
    4-> 5 -> 7 ->NULL
```

分析

对于任意一棵树来说，可以分成下面三种情况

1. 若左子树存在，右子树也存在，左子树的next是右子树，右子树的next是父节点的兄弟节点的第一个孩子（root的大侄子）；
2. 若左子树存在，右子树不存在，左子树的next是父节点的兄弟节点的第一个孩子；
3. 若左子树不存在，右子树存在，右子树的next是父节点的兄弟节点的第一个孩子；

所以问题的关键是寻找root的大侄子，依次查看root->next是否存在左右子树，否则，root = root->next;

同样，可以通过递归操作完成。

```
/**
 * Definition for binary tree with next pointer.
 * struct TreeLinkNode {
 *  int val;
 *  TreeLinkNode *left, *right, *next;
 *  TreeLinkNode(int x) : val(x), left(NULL), right(NULL), next(
NULL) {}
 * };
 */
class Solution {
public:
    void connect(TreeLinkNode *root) {
        if(!root) return;
        TreeLinkNode *temp = root->next;
        while(temp){
            if(temp->left){
                temp = temp->left;
                break;
            }else if(temp->right){
                temp = temp->right;
                break;
            }
            temp = temp->next;
        }
        if(root->right) root->right->next = temp;
        if(root->left) root->left->next = (root->right)?(root->r
ight):(temp);
        connect(root->right);
        connect(root->left);
    }
};
```

129. Sum Root to Leaf Numbers

直达：https://leetcode.com/problems/sum-root-to-leaf-numbers/description/

Given a binary tree containing digits from `0-9` only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123` .

Find the total sum of all root-to-leaf numbers.

For example,

```
    1
   / \
  2   3
```

The root-to-leaf path `1->2` represents the number `12` .
The root-to-leaf path `1->3` represents the number `13` .

Return the sum = 12 + 13 = `25` .

分析

类似于Q112, 不同之处在于去路径之和的时候要对之前的路径乘以10。

C++代码

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int sumNumbers(TreeNode* root) {
        int res = 0;
        if(!root) return 0;
        helper(root, root->val, res, 0);
        return res;
    }
private:
    void helper(TreeNode* root, int path, int& res, int height){
        if (!root->left && !root->right){
            res += path;
            return;
        }else{
            if(root->left) helper(root->left, 10 * path + root->
left->val, res, height+1);
            if(root->right) helper(root->right, 10 * path + root
->right->val, res, height+1);
        }
    }
};
```

144. Binary Tree Preorder Traversal

Given a binary tree, return thepreordertraversal of its nodes' values.

For example:
Given binary tree `[1,null,2,3]` ,

```
   1
    \
     2
    /
   3
```

return `[1,2,3]` .

Note: Recursive solution is trivial, could you do it iteratively?

分析

用递归解决二叉树的遍历是非常普遍的解法，题目提出能否使用迭代来进行二叉树的先序遍历。

对于先序遍历，需要使用堆栈做辅助。首先树顶入栈，然后执行while循环直到堆栈为空。在循环中每次弹出栈顶元素，将结果保存在数组中然后再将弹出节点的右子树和左子树一次入栈。首先，例如树

```
      1
    /   \
   2     3
  / \   / \
 4   5 6   7
```

1. 入栈，执行while循环.
2. 弹出1，将3，2入栈。此时res=[1], stack=[3,2]
3. 弹出2，将5，4入栈。此时res=[1, 2], stack=[3,5,4]

4. 弹出4。res=[1,2,4], stack=[3,5]
5. 弹出5。res=[1,2,4,5]. stack=[3]
6. 弹出3，将7，6入栈。res=[1,2,4,5,3], stack=[7,6]
7. 弹出6。res=[1,2,4,5,3,6], stack=[7]
8. 弹出7。res=[1,2,4,5,3,6,7], stack=[]
9. 栈为空，循环结束，返回res。

C++算法

递归

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> res;
        helper(root, res);
        return res;
    }
private:
    void helper(TreeNode* root, vector<int>& res){
        if (!root) return;
        res.push_back(root->val);
        if (root->left) helper(root->left, res);
        if (root->right) helper(root->right, res);
    }
};
```

迭代

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> res;
        if(!root) return res;
        stack<TreeNode*> stk;
        stk.push(root);
        while(!stk.empty()){
            TreeNode* temp = stk.top();
            stk.pop();
            res.push_back(temp->val);
            if (temp -> right) stk.push(temp->right);
            if (temp -> left) stk.push(temp->left);
        }
        return res;
    }
};
```

89. Gray Code

直达：https://leetcode.com/problems/gray-code/description/

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integernrepresenting the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, givenn= 2, return `[0,1,3,2]`. Its gray code sequence is:

```
00 - 0
01 - 1
11 - 3
10 - 2
```

Note:
For a givenn, a gray code sequence is not uniquely defined.

For example, `[0,2,3,1]` is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

分析：

当n=1时，输出初始化为 `[0,1]` ,n=2时，交替的向尾部添加 `[0,1], [1,0]` (奇数位置填 `[0,1]` , 偶数位置填 `[1,0]`). 结果即是 `[00, 10, 11, 01]` 。尾部加0就是乘2，尾部加一就是乘2再加1.

以此类推, 第k个在第k-1个的数组的每个数的基础上在尾部分别交替添加 `[0,1], [1,0]` 。

因此，迭代的进行更新数组即可。

算法：

```
class Solution {
public:
    vector<int> grayCode(int n) {
        vector<int> res;
        res.push_back(0);
        if(n == 0) return res;
        res.push_back(1);
        for(int i=1; i<n; i++){
            res = helper(res, i);
        }
        return res;
    }
private:
    vector<int> helper(vector<int>& nums, int k){
        vector<int> res;
        int add_num = pow(2, k);
        for(int i=0; i<nums.size(); i++){
            if(i%2 == 0){
                res.push_back(nums[i]*2);
                res.push_back(nums[i]*2+1);
            }else{
                res.push_back(nums[i]*2+1);
                res.push_back(nums[i]*2);
            }
        }
        return res;
    }
};
```

136. Single Number

直达：https://leetcode.com/problems/single-number/description/

Given an array of integers, every element appearstwiceexcept for one. Find that single one.

Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

分析

在数组中查找只出现一次的数字，其它数字出现两次。遍历数组，一次将遍历到的数执行亦或，相同的数字通过两次亦或结果是0，遍历之后剩下的那个数就是只出现一次的那个数。

例如数组[4, 4, 1, 3, 3] 对应的二进制依次是[100, 100, 001, 011, 011], 按位依次亦或得到的结果是[0, 0, 1], 即对应只出现一次的1的二进制。ß

C++代码

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int res = 0;
        for(int i = 0; i<nums.size(); i++){
            res ^= nums[i];
        }
        return res;
    }
};
```

137. Single Number II

直达：https://leetcode.com/problems/single-number-ii/description/

Given an array of integers, every element appearsthreetimes except for one, which appears exactly once. Find that single one.

Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

分析

和Q136类似，对于一个数组，出现3次的数字取和摩3之后结果是0，依次遍历，按位相加后摩3，得到的结果便是只出现一次的那个数。例如数组[3, 3, 5, 3, 4, 5, 5]对应的二进制编码是[011, 011, 101, 011, 100, 101, 101], 按位取和之后是[4, 3, 6]，摩3之后是[1, 0, 0]. 对应的二进制是4，即只出现一次的那个数。

每一位的运算符合下面规则：

00 + 0 -> 00, 01 + 0 -> 01, 10 + 0 -> 10

00 + 1 -> 01, 01 + 1 -> 10, 10 + 1 -> 00

设二进制高位为a, 低位是b，遍历到的数是num，对应的真值表是

输入**a**	输入**b**	**num**	输出**a**	输出**b**	**a ^ num**	**b ^ num**
0	0	0	0	0	0	0
0	1	0	0	1	0	1
1	0	0	1	0	1	0
0	0	1	0	1	1	1
0	1	1	1	0	1	0
1	0	1	0	0	0	1

输入a, 输入b^num，输出b对应的真值表是

输入**a**	输入**b^num**	输出**b**
0	0	0
0	1	1
1	0	0
1	1	0

于是有

输出b = ~输入a & (输入b^num)

输出b，输入a^num， 输出a对应的眞值表是

输出**b**	输入**a^num**	输出**a**
0	0	0
0	1	1
1	0	0
1	1	0

于是有

输出a = ~输出b&(输入a^num)

于是遍历的更新规则是

b = ~a&(b^num)

a = ~b&(a^num)

按照算法来说，得到的结果应该是出现1次的那个数，直接返回b即可。

C++代码

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int a = 0;
        int b = 0;
        for (auto num : nums){
            b = (num^b) & ~a;
            a = (num^a) & ~b;
        }
        return b;
    }
};
```

191. Number of 1 Bits

直达：https://leetcode.com/problems/number-of-1-bits/description/

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as theHamming weight).

For example, the 32-bit integer '11' has binary representation `00000000000000000000000000001011`, so the function should return 3.

Credits:
Special thanks to@tsfor adding this problem and creating all test cases.

分析：

二进制中，n&(n-1)作用是将n的二进制表示中的最低位为1的改为0，每更改一次计数器+1，知道n=0停止。

C++代码

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

191. Reverse Bits

直达：https://leetcode.com/problems/reverse-bits/description/

Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596 (represented in binary as**00000010100101000001111010011100**), return 964176192 (represented in binary as**00111001011110000010100101000000**).

Follow up:
If this function is called many times, how would you optimize it?

Related problem:Reverse Integer

分析

n&1 可以取n的最低位，取完之后右移一位，并将取得的数添加到res的右侧（即右移1位之后再加1）

C++代码

```
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t res = 0;
        for(int i = 0; i < 32; i++){
            uint32_t bit = n & 1;
            n = n >> 1;
            res = res << 1;
            res += bit;
        }
        return res;
    }
};
```

5. Longest Palindromic Substring

直达：https://leetcode.com/problems/longest-palindromic-substring/description/

Given a string**s**, find the longest palindromic substring in**s**. You may assume that the maximum length of**s**is 1000.

Example:

```
Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example:

```
Input: "cbbd"
Output: "bb"
```

分析

遍历字符串，从遍历到的字符开始向两边查找，判断是否满足回文。注意需要区分奇数子串和偶数子串两种情况。

C++代码

```
class Solution {
public:
    string longestPalindrome(string s) {
        int res_left = 0;
        int res_right = 0;
        if (s.size() < 2)  return s;
        for(int i=0; i<=s.size()-1;i++){
            // 奇数长度子串
            int left = i-1;
            int right = i+1;
            while(s[left] == s[right] && left >= 0 && right <= s
.size()-1){
                left--; right++;
            }
            if((right-left-2) > (res_right-res_left)){
                res_right = right-1;
                res_left = left+1;
            }
            // 偶数长度子串
            right = i+1;
            left = i;
            while(s[left] == s[right] && left >= 0 && right <= s
.size()-1){
                left--; right++;
            }
            if((right-left-2) > (res_right-res_left)){
                res_right = right-1;
                res_left = left+1;
            }
        }
        return s.substr(res_left, (res_right-res_left+1));
    }
};
```

14. Longest Common Prefix

直达：https://leetcode.com/problems/longest-common-prefix/description/

Write a function to find the longest common prefix string amongst an array of strings.

分析

没有什么难度的一道题，两层循环对比公共前缀即可。

C++代码

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (strs.size() == 0) return "";
        if (strs.size() == 1) return strs[0];
        string res="";
        for(int j = 0; j<INT_MAX; j++){
            for(int i = 1; i < strs.size(); i++){
                if(strs[0].size() <= j || strs[i].size() <= j ||
 strs[i][j] != strs[0][j])    return res;
            }
            res += strs[0][j];
        }
        return res;
    }
};
```

125. Valid Palindrome

直达：https://leetcode.com/problems/valid-palindrome/description/

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,
`"A man, a plan, a canal: Panama"` is a palindrome.
`"race a car"` isnota palindrome.

Note:
Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrom

分析

首先将字符串清洗一下，包括两个步骤：

1. 保留字母和数字
2. 将大写字母转小写

然后使用头尾两个指针判断字符串是否是回文。

C++代码

```
class Solution {
public:
    bool isPalindrome(string s) {
        string str;
        for(int i = 0; i< s.size(); i++){
            if( (s[i]>='a' && s[i]<='z') || (s[i]>='0' && s[i]<=
'9')) str+=s[i];
            if(s[i]>='A' && s[i]<='Z') str+= 'a'+(s[i]-'A');
        }
        for(int i=0, j = str.size()-1; i<=j;i++,j--){
            if(str[i]!=str[j]) return false;
        }
        return true;
    }
};
```

85. Maximal Rectangle

直达: https://leetcode.com/problems/maximal-rectangle/description/

Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return its area.

For example, given the following matrix:

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

Return 6.

给出一个含有1和0的二位矩阵，返回用1组成的最大矩形区域的面积

解析

该问题是由 Q84 衍生而来，将问题还原为Q84的方法是将矩阵的每一行都看成一个Q84的问题，其中bar的高度是从matrix[i][j]开始，从下往上数不间断的1的个数。显然可以通过动态规划构建。

例如上图对应的矩阵是

```
1 0 1 0 0
2 0 2 1 1
3 1 3 2 2
4 0 0 3 0
```

C++实现

```
class Solution {
public:
    int maximalRectangle(vector<vector<char>>& matrix) {
```

```
        int row = matrix.size();
        if(row == 0) return 0;
        int col = matrix[0].size();
        int res = 0;
        vector<vector<int>> dp(row, vector<int>(col, 0));
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(i > 0) dp[i][j] = (matrix[i][j]-'0') * (dp[i-1
][j] + (matrix[i][j]-'0'));
                else dp[i][j] = matrix[i][j]-'0';
            }
        }

        for(int i = 0; i < row; i++){
            res = max(res, largestRectangleArea(dp[i]));
        }
        return res;
    }
private:
    int largestRectangleArea(vector<int>& heights) {
        stack<int> stk;
        heights.push_back(0);
        int res = 0;
        for (int i = 0; i < heights.size(); i++){
            if(stk.empty() || heights[i] > heights[stk.top()]){
                stk.push(i);
            }else{
                int h = stk.top();
                stk.pop();
                res = max(res, (stk.empty()?i:i-stk.top()-1) * h
eights[h]);
                i--;
            }
        }
        return res;
    }
};
```

91. Decode Ways

直达：https://leetcode.com/problems/decode-ways/description/

A message containing letters from `A-Z` is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,
Given encoded message `"12"`, it could be decoded as `"AB"` (1 2) or `"L"` (12).

The number of ways decoding `"12"` is 2.

分析：

当遍历到字符串 s 的第k个位置时，对应的路径的数量取决于第k-1和k-2个位置的内容和路径数。所以要采用动态规划求解，具体要分成下面几种情况：

1. `s[i] == 0`
   i. 如果 `0<(s[i-1]-'0')*10+s[i]-'0'<=26`，则第k个位置的路径数等于第k-2个位置的路径数，例如 `*10*` ，要从左侧 `*` 处断开；
   ii. 反之，路径数为0，例如 `*30*` .无法继续，路径数为0，直接返回0。
2. `s[i] != 0`
   i. 如果 `s[i-1]==0`
      i. 如果 `0<(s[i-1]-'0')*10+s[i]-'0'<=26` ，则第k个位置的路径数等于第k-1个位置的路径数；
      ii. 否则，路径数为0，直接返回，如 `*00*` 这种情况

ii. 如果 `s[i-1] != 0`

i. 如果 `0<(s[i-1]-'0')*10+s[i]-'0'<=26` ，则第k个位置的路径数等于第k-1个位置的路径数加上k-2个位置的路径数，如 `*123*` ，i指向3，断成 `*12,3*` 或者 `*1,23*` 都可以；

ii. 否则，则第k个位置的路径数等于第k-1个位置的路径数，如 `*128*` 只能断成 `*12,8*` 。

C++代码：

```
class Solution {
public:
    int numDecodings(string s) {
        int n = s.size();
        if(n == 0) return 0;
        if(s[0] == '0') return 0;
        vector<int> dp(n+1, 0);
        dp[0] = 1;
        dp[1] = 1;
        for(int i = 1; i < n; i++){
            int temp = ((s[i-1]-'0')*10+s[i]-'0');
            if(s[i] == '0'){
                if (0 < temp && 26 >= temp) dp[i+1] = dp[i-1];
                else return 0;
            }else{
                if(s[i-1] != '0'){
                    if (0 < temp && 26 >= temp) dp[i+1] = dp[i]
+ dp[i-1];
                    else dp[i+1] = dp[i];
                }else{
                    if (0 < temp && 26 >= temp) dp[i+1] = dp[i];
                    else return 0;
                }
            }
        }
        return dp[n];
    }
};
```

121. Best Time to Buy and Sell Stock

直达：https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/

Say you have an array for which theithelement is the price of a given stock on dayi.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Example 1:

```
Input: [7, 1, 5, 3, 6, 4]
Output: 5

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs t
o be larger than buying price)
```

Example 2:

```
Input: [7, 6, 4, 3, 1]
Output: 0

In this case, no transaction is done, i.e. max profit = 0.
```

分析

保存两个变量

1. minPrice：当前遍历到的最少买入价格；
2. maxProfit: 最大收益；

则更新公式为：

1. minPrice = min(prices[i], minPrice);
2. maxProfit = max(maxProfit, prices[i]-minPrice);

C++代码

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        if(prices.size() <= 0) return 0;
        int minPrice = prices[0];
        int maxProfit = 0;
        for(int i=0; i< prices.size(); i++){
            minPrice = min(prices[i], minPrice);
            maxProfit = max(maxProfit, prices[i] - minPrice);
        }
        return maxProfit;
    }
};
```

Python代码

```python
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        res = 0
        if (len(prices) <= 1):
            return 0
        minPrice = prices[0]
        maxProfit = 0
        for price in prices:
            minPrice = min(minPrice, price)
            maxProfit = max(maxProfit, price-minPrice)
        return maxProfit
```

198. House Robber

直达：https://leetcode.com/problems/house-robber/description/

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Credits:
Special thanks to@ifanchufor adding this problem and creating all test cases. Also thanks to@tsfor adding additional test cases.

分析

动态规划的经典题目，动态规则也非常简单。第i天盗得的总和是第i-2天，或者第i-3天最大值加上nums[i]， 所以有动态规划更新规则：

```
dp[i+1] = max(dp[i-1], dp[i-2]) + nums[i];
```

注：为了编码方便，动态规划数组添加了一个0的头。

C++代码

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int len = nums.size();
        if(len == 0) return 0;
        vector<int> dp(len+1, 0);
        dp[1] = nums[0];
        for (int i = 1; i < len; i++){
            if (i == 1) dp[i+1] = dp[0] + nums[i];
            else dp[i+1] = max(dp[i-1], dp[i-2]) + nums[i];
        }
        return max(dp[len], dp[len-1]);
    }
};
```

17. Letter Combinations of a Phone Number

Given a digit string, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below.


```
Input:Digit string "23"
Output:["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

Note:
Although the above answer is in lexicographical order, your answer could be in any order you want.

分析

递归求解。

C++代码

```
class Solution {
public:
    void letterCombinations_helper(vector<string>&res, string di
gits, int idx, string path){
        vector<string> phone = {"abc", "def", "ghi", "jkl", "mno"
, "pqrs", "tuv", "wxyz"};
        string str = phone[digits[idx]-'2'];
        for(int i=0; i<str.size(); i++){
            if(idx == digits.size() -1){
                res.push_back(path+str[i]);
            }else{
                letterCombinations_helper(res, digits, idx+1, pa
th+str[i]);
            }
        }
    }
    vector<string> letterCombinations(string digits) {
        vector<string> res;
        int idx = 0;
        string path = "";
        letterCombinations_helper(res, digits, idx, path);
        return res;
    }
};
```

7.8 Subsets

直达: https://leetcode.com/problems/subsets/description/

Given a set of**distinct**integers,nums, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

For example,
If **nums** = `[1,2,3]` , a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

分析

求子集是递归的典型题目之一，每次向路径中添加一个数即可。

C++代码

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> res;
        vector<int> path;
        helper(res, path, nums, 0, 0);
        return res;
    }
private:
    void helper(vector<vector<int>>& res, vector<int>& path, vec
tor<int> nums, int pos, int cnt){
        if(cnt <= nums.size()){
            res.push_back(path);
        }
        for(int i = pos; i<nums.size(); i++){
            path.push_back(nums[i]);
            helper(res, path, nums, i, cnt+1);
            path.pop_back();
        }
    }
};
```

87. Scramble String

直达: https://leetcode.com/problems/scramble-string/description/

Given a string s1, we may represent it as a binary tree by partitioning it to two non-empty substrings recursively.

Below is one possible representation of s1= `"great"` :

```
    great
   /    \
  gr    eat
 / \    /  \
g   r  e   at
           / \
          a   t
```

To scramble the string, we may choose any non-leaf node and swap its two children.

For example, if we choose the node `"gr"` and swap its two children, it produces a scrambled string `"rgeat"` .

```
    rgeat
   /    \
  rg    eat
 / \    /  \
r   g  e   at
           / \
          a   t
```

We say that `"rgeat"` is a scrambled string of `"great"` .

Similarly, if we continue to swap the children of nodes `"eat"` and `"at"` , it produces a scrambled string `"rgtae"` .

```
    rgtae
   /    \
  rg    tae
 / \    /  \
r   g  ta  e
       / \
      t   a
```

We say that `"rgtae"` is a scrambled string of `"great"` .

Given two strings s1 and s2 of the same length, determine if s2 is a scrambled string of s1.

分析

将s1分成两个子字符串s11和s12, 则无论怎么变化, s11和s12的所有的字符串都在s2的相同的子树里，或者左子树s21，或者右子树s12。

如果s11与s21并且s12与s22都满足Scramble或者s11与s22并且s12与s21都满足Scramble，则s1与s2满足Scramble。显然，需要通过递归求解。

C++代码

```
class Solution {
public:
    bool isScramble(string s1, string s2) {
        if(s1 == s2) return true;
        if(s1.size() != s2.size()) return false;
        string str1 = s1; string str2 = s2;
        sort(str1.begin(), str1.end());
        sort(str2.begin(), str2.end());
        if (str1 != str2) return false;
        for(int i = 1; i < str1.size(); i++){
            string s11 = s1.substr(0, i);
            string s12 = s1.substr(i);
            string s21 = s2.substr(0,i);
            string s22 = s2.substr(i);
            if(isScramble(s11, s21) && isScramble(s12, s22)) ret
urn true;
            s21 = s2.substr(s1.size()-i);
            s22 = s2.substr(0, s1.size()-i);
            if(isScramble(s11, s21) && isScramble(s12, s22)) ret
urn true;
        }
        return false;
    }
};
```

90. Subsets II

直达: https://leetcode.com/problems/subsets-ii/description/

Given a collection of integers that might contain duplicates,**nums**, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

For example,
If **nums=** `[1,2,2]` , a solution is:

```
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

分析:

是Q78的扩展，不同之处在于输入的是含有重复元素的数组。

处理方法是在插入元素的时候，略过和它前面内容相同的元素。

C++代码

```cpp
class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        vector<vector<int>> res;
        vector<int> path;
        sort(nums.begin(), nums.end());
        helper(res, path, nums, 0, 0);
        return res;
    }
private:
    void helper(vector<vector<int>>& res, vector<int>& path, vec
tor<int> nums, int pos, int cnt){
        if(cnt <= nums.size()){
            res.push_back(path);
            for(int i = pos; i<nums.size(); i++){
                if(i!=pos && nums[i] == nums[i-1]) continue;
                path.push_back(nums[i]);
                helper(res, path, nums, i+1, cnt+1);
                path.pop_back();
            }
        }
    }
};
```

122. Best Time to Buy and Sell Stock II

直达：https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/description/

Say you have an array for which the i^{th} element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

分析

只要这一天比前一天股票价格高，就会有收入，因此可以使用贪心算法。

C++代码

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int res = 0;
        if(prices.size() > 0) int minPrice = prices[0];
        for(int i = 1; i<prices.size(); i++){
            if(prices[i] - prices[i-1] > 0) res += prices[i] - p
rices[i-1];
        }
        return res;
    }
};
```

6. ZigZag Conversion

直达：https://leetcode.com/problems/zigzag-conversion/description/

The string `"PAYPALISHIRING"` is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: `"PAHNAPLSIIGYIR"`

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string text, int nRows);
```

`convert("PAYPALISHIRING", 3)` should return `"PAHNAPLSIIGYIR"`

分析

这道题更像是一道数学题，关键是找到返回字符串的字符位置和原字符串的字符位置的对应关系。0-n的整数数组（字符串下标）更容易看到规律，例如numRows = 5

对应Z形是

```
0       8           16
1     7 9         15 17
2   6   10    14    18    ...
3 5     11 13       19 21
4       12          20
```

两个竖的间距是2*(numRows-1)，在第i行(i>=1 && i<=numRows-2)，中间要插入一个间距为2*(numRows-i)的数字。

换成字符串时，注意下标不要超过字符串的长度。

C++代码

```
class Solution {
public:
    string convert(string s, int numRows) {
        if(s.size() < 3) return s;
        if(numRows <= 1) return s;
        string res;
        for (int i = 0; i <= numRows-1; i++){
            int j = i;
            while(j < s.size()){
                res+=(s[j]);
                if(j + 2*(numRows-1-i) < s.length() && i > 0 &&
i < numRows - 1){
                    res+=(s[j + 2*(numRows-1-i)]);
                }
                j += 2 * (numRows-1);
            }
        }
        return res;
    }
};
```

7. Reverse Integer

直达：https://leetcode.com/problems/reverse-integer/description/

iven a 32-bit signed integer, reverse digits of an integer.

Example 1:

```
Input: 123
Output: 321
```

Example 2:

```
Input: -123
Output: -321
```

Example 3:

```
Input: 120
Output: 21
```

Note:
Assume we are dealing with an environment which could only hold integers within the 32-bit signed integer range. For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

分析

保存正负号后使用绝对值进行运算。

使用 %10取个位，使用 /10进行右移1位。

使用长整型保存结果（Python不用）以便进行溢出判断。

代码

C++

```
class Solution {
public:
    int reverse(int x) {
        if (x == INT_MIN) return 0;
        int sign = x>0?1:-1;
        x = sign * x;
        double res = 0;
        while(x != 0){
            res *= 10;
            res += x%10;
            x /= 10;
        }
        if ( (sign > 0 && res > INT_MAX) || (sign < 0 && res-1 >
 INT_MAX) ) return 0;
        else return sign * int(res);
    }
};
```

python

```
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        pos_x = abs(x)
        res = 0
        while(pos_x > 0):
            res *= 10
            res += pos_x%10
            pos_x = pos_x/10
        if(x < 0):
            res = -res
        if abs(res) > 0x7FFFFFFF:
            return 0
        else:
            return res
```

9. Palindrome Number

直达：https://leetcode.com/problems/palindrome-number/description/

Determine whether an integer is a palindrome. Do this without extra space.

分析

首先取得整数的长度len。

使用x%10取低位，x/pow(10, len-1)取高位，循环判断直到x到个位。

代码

C++

```
class Solution {
public:
    bool isPalindrome(int x) {
        if (x < 0) return false;
        if (x < 10) return true;
        int temp_x = x;
        int x_len = 0;
        while(temp_x != 0){
            x_len++;
            temp_x /= 10;
        }
        while(x_len > 1 && x%10 == int(x/pow(10, x_len-1))){
            x -= int(x/pow(10,x_len-1)) * pow(10,x_len-1);
            x /= 10;
            x_len -= 2;
        }
        if (x_len < 2) return true;
        else return false;
    }
};
```

168. Excel Sheet Column Title

直达：https://leetcode.com/problems/excel-sheet-column-title/description/

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB
```

分析

这道题是十进制转26进制，但是注意十进制是从1开始计数，所以开始的时候n要减1。

例如26 -> Z， (26-1)%26 = 25, 25+'A'对应的是Z。

C++代码

```cpp
class Solution {
public:
    string convertToTitle(int n) {
        string res;
        char temp;
        while(n){
            n -= 1;
            int bit = n%26;
            temp = bit + 'A';
            res = temp + res;
            n /= 26;
        }
        return res;
    }
};
```

171. Excel Sheet Column Number

直达：https://leetcode.com/problems/excel-sheet-column-number/description/

Related to question Excel Sheet Column Title

Given a column title as appear in an Excel sheet, return its corresponding column number.

For example:

```
    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28
```

Credits:
Special thanks to@tsfor adding this problem and creating all test cases.

分析

这道题本质上是二十六进制转十进制

C++代码

```cpp
class Solution {
public:
    int titleToNumber(string s) {
        int len = s.size();
        int res = 0;
        for (int i = 0; i < len; i++){
            res += pow(26, i) * (s[len-1-i]-'A'+1);
        }
        return res;
    }
};
```

15. Three Sum

直达：https://leetcode.com/problems/3sum/description/

Given an arraySofnintegers, are there elementsa,b,cinSsuch thata+b+c= 0? Find all unique triplets in the array which gives the sum of zero.

Note: The solution set must not contain duplicate triplets.

```
For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

分析

可以将该问题转化成Q1，依次将数组的元素作为-target。也可以排序后确定两个值，然后使用二分查找需要的那个值。下面只实现第二个方法。

C++代码

```
class Solution {
public:
    bool binary_search(vector<int>& nums, int target, int left,
int right){
        int mid = 0;
        while(left <= right){
            mid = left + (right - left) /2;
            if(nums[mid] < target) left = mid+1;
            else if (nums[mid] > target) right = mid-1;
            else return true;
        }
        if (nums[mid] == target) return true;
        else return false;
    }
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> res;
        if (nums.size() <= 2) return res;
        sort(nums.begin(), nums.end());
        for(int i = 0; i <= nums.size() - 3; i++){
            for(int j = i+1; j <= nums.size() - 2; j++){
                if(binary_search(nums, -(nums[i] + nums[j]), j+1
, nums.size()-1)){
                    res.push_back({nums[i], nums[j], -(nums[i]+n
ums[j])});
                }
                while(nums[j+1] == nums[j]) j++;
            }
            while(nums[i+1] == nums[i]) i++;
        }
        return res;
    }
};
```

167. Two Sum II - Input array is sorted

直达：https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/description/

Given an array of integers that is already**sorted in ascending order**, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would haveexactlyone solution and you may not use thesameelement twice.

Input:numbers={2, 7, 11, 15}, target=9
Output:index1=1, index2=2

分析

使用两个指针，一个从左向右查找，一个从右向左查找，不同于O(n)方法的每次移动一个值。可以采用二分查找更新到最近的那个值。这样复杂度便是O(logn).

C++代码

```
class Solution {
private:
    int binarySearch(vector<int> nums, int target, int left, int
 right){
        int mid = (right-left)/2 + left;
        while(left < right){
            if(nums[mid] == target) return mid;
            else if(nums[mid] < target) left = mid+1;
            else right = mid-1;
            mid = (right-left)/2 + left;
        }
        return mid;
    }
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        int left = 0;
        int right = numbers.size()-1;
        vector<int> res;
        while(left <= right){
            if (numbers[left] + numbers[right] == target){
                res.push_back(left+1);
                res.push_back(right+1);
                return res;
            }else if(numbers[left] + numbers[right] < target){
                left = binarySearch(numbers, target-numbers[righ
t], left+1, right-1);
            }else{
                right = binarySearch(numbers, target-numbers[lef
t], left+1, right-1);
            }
        }
    }
};
```

169. Majority Element

直达：https://leetcode.com/problems/majority-element/description/

Given an array of sizen, find the majority element. The majority element is the element that appears**more than** `⌊ n/2 ⌋` times.

You may assume that the array is non-empty and the majority element always exist in the array.

分析

本题很容易想到基于排序或者哈希表的方法，但是排序的方法的时间复杂度是O(nlogn)，而哈希表需要占用额外的存储空间，若要使用O(n)且不占用额外存储空间的方法，可以使用Boyer-Moore投票方法

Boyer-Moore投票方法是每次删除两个不同的数，直到最后剩下的一定是出现次数大于$\lfloor n/2 \rfloor$的数，但是数组的删除是非常困难的，因为每删除一个元素都要移动数组，所以可以采用计数器的方式。

设当前统计的数是flag，遇到等于flag的数count+1，否则count - 1，如果count < 0， 则表示该数不一定是要返回的数，则可以换一个数。

C++代码

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