001. Two Sum [Easy]

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.

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

Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1]. UPDATE (2016/2/13): The return format had been changed to zero-based indices. Please read the above updated description carefully.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        vector<int> v(2);
        for(int i = 0; i < nums.size(); i++) {
            for(int j = i + 1; j < nums.size(); j++) {
                if(nums[i] + nums[j] == target) {
                    v[0] = i;
                    v[1] = j;
                    return v;
                }
            }
        }
    }
};
```

002. Add Two Numbers [Medium]

You are given two linked lists representing two non-negative numbers. 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.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4) Output: 7 -> 0 -> 8

```
/**
 * 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) {
        ListNode *h = new ListNode(0);
        ListNode *p = h;
        int temp = 0;
        while(l1 != NULL && l2 != NULL) {
            p->next = new ListNode((temp + l1->val + l2->val) % 10);
            temp = (temp + l1->val + l2->val) / 10;
            p = p->next;
            l1 = l1->next;
            l2 = l2->next;
        }
        while(l1 != NULL) {
            p->next = new ListNode((temp + l1->val) % 10);
            temp = (temp + l1->val) / 10;
            p = p->next;
            l1 = l1->next;
        }
        while(l2 != NULL) {
            p->next = new ListNode((temp + l2->val) % 10);
            temp = (temp + l2->val) / 10;
            p = p->next;
            l2 = l2->next;
        }
        if(temp) {
            p->next = new ListNode(1);
        }
        return h->next;
    }
};
```

003. Longest Substring Without Repeating Characters [Medium]

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 a subsequence and not a substring.

分析：用两个指针ij从头开始遍历字符串。ij分别表示最长字串的首尾下标。如果第j个与i与j当中的某处k重复，那么只需i从k+1开始继续判断是否有重复的就好。当然，在i++一直到k的过程中，不要忘记把已经收录的字符存在标记为不存在。所以用一个book数组标记该字符在i~j当中是否出现过。每一次找到重复的字符的时候判断j-i是否比最大值大。一个特例是，如果i~j中一直让j到了最后一个字符都没有重复但是此时的j-i是最长的长度，所以要在return语句前再加上一句判断j-i的大小是否比当前maxlen大。因为i和j都只遍历了字符串一次，所以时间复杂度为O(n)。

```
class Solution {
public:
```

```
    int lengthOfLongestSubstring(string s) {
        int i = 0, j = 0, len = s.length();
        int maxlen = 0;
        int book[256] = {0};
        while(j < len) {
            if(book[s[j]] == 1) {
                maxlen = max(maxlen, j - i);
                while(s[i] != s[j]) {
                    book[s[i]] = 0;
                    i++;
                }
                i++;
            } else {
                book[s[j]] = 1;
            }
            j++;
        }
        maxlen = max(maxlen, j - i);
        return maxlen;
    }
};
```

005. Longest Palindromic Substring [Medium]

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”

题目大意：给一个字符串，找它的回文子串中长度最长的一个子串～

分析：首先令result为第一个元素，如果没有第一个元素就返回空字串，遍历字串，对于每个元素都从中间向两边寻找是否有回文子串，返回这个子串，将它的长度与result比较，如果比result长就更新result～ 要分为奇数还是偶数考虑，奇数就左右都是i，偶数就是左边是i右边是i+1然后扩展～

```
class Solution {
public:
    string longestPalindrome(string s) {
        int len = s.length();
        if (len == 0) return "";
```

```
        string result = s.substr(0, 1);
        for (int i = 0; i <= len - 2; i++) {
            string temp = midToSide(s, i, i);
            if (temp.length() > result.length())
                result = temp;
            temp = midToSide(s, i, i + 1);
            if (temp.length() > result.length())
                result = temp;
        }
        return result;
    }

    string midToSide(string s, int left, int right) {
        while (left >= 0 && right <= s.length() - 1 && s[left] ==
s[right]) {
            left--;
            right++;
        }
        return s.substr(left + 1, right - left - 1);
    }
};
```

006. ZigZag Conversion [Easy]

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”.

```
class Solution {
public:
    string convert(string s, int numRows) {
        string t[numRows];
        string ans;
        int i = 0;
        while(i < s.length()) {
            for(int j = 0; j < numRows && i < s.length(); j++)
                t[j] += s[i++];
            for(int k = numRows - 2; k > 0 && i < s.length(); k--)
                t[k] += s[i++];
        }
        for(int j = 0; j < numRows; j++) {
            ans += t[j];
        }
```

```
        return ans;
    }
};
```

007. Reverse Integer [Easy]

Example1: x = 123, return 321 Example2: x = -123, return -321

Have you thought about this? Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

Update (2014-11-10): Test cases had been added to test the overflow behavior.

```
class Solution {
public:
    int reverse(int x) {
        string s = to_string(x);
        if(s[0] == '-')
            std::reverse(s.begin() + 1, s.end());
        else
            std::reverse(s.begin(), s.end());
        long long int temp = stoll(s);
        if(temp > 2147483647 || temp < -2147483648)
            return 0;
        return (int)temp;
    }
};
```

008. String to Integer (atoi) [Easy]

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Update (2015-02-10): The signature of the C++ function had been updated. If you still see your function signature accepts a const char * argument, please click the reload button to reset your code definition.

spoilers alert... click to show requirements for atoi.

Requirements for atoi: The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned. If the correct value is out of the range of representable values, INT_MAX (2147483647) or INT_MIN (-2147483648) is returned.

```
class Solution {
public:
    int myAtoi(string str) {
        if(str.length() == 0)
            return 0;
        int i = 0, flag = 1;
        while(isspace(str[i])) {
            i++;
        }
        if(str[i] == '+' || str[i] == '-') {
            if(str[i] == '-')
                flag = 0;
            i++;
        }
        str = str.substr(i);
        for(int j = 0; j < str.length(); j++) {
            if(!isdigit(str[j])) {
                str = str.substr(0, j);
                break;
            }
        }
        if(str.length() == 0)
            return 0;
        if(str.length() > 10) {
            if(flag == 0)
                return -2147483648;
            else
                return 2147483647;
        }
        long long int ans = stoll(str);
        if(flag == 0)
            ans = 0 - ans;
```

```
        if(ans > 2147483647) {
            return 2147483647;
        } else if (ans < -2147483648) {
            return -2147483648;
        } else {
            return (int)ans;
        }
    }
};
```

009. Palindrome Number [Easy]

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

Some hints: Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem “Reverse Integer”, you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

```
class Solution {
public:
    bool isPalindrome(int x) {
        if(x < 0)
            return false;
        int len = 1;
        while(x / len >= 10) {
            len = len * 10;
        }
        while(x) {
            int left = x / len;
            int right = x % 10;
            if(left != right) {
                return false;
            }
            x = x % len;
            x = x / 10;
            len = len / 100;
        }
        return true;
    }
};
```

011. Container With Most Water [Medium]

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) 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.

题目大意：给出一个数组height, height[i] = j表示横坐标为i处的高为j，以(i, j)与x轴作垂线段，计算两条垂线段和x轴组成的容器能装的最多的水的容量是多少~

分析：容器两条边中取最短的那条边为影响容器容积的高，所以说，我们先假设left和right分别在最左边最右边，要想求得容器容积的最大值，需要考虑改变最短边的高度，如果左边短就让left++，如果右边短就让right–，直到直到一个area容积最大的值返回～

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

012. Integer to Roman [Medium]

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

题目大意：将一个整数转化为罗马数字表示法～

分析：根据罗马数字的表示方法，我们直到罗马字符有”M”, “CM”, “D”, “CD”, “C”, “XC”, “L”, “XL”, “X”, “IX”, “V”, “IV”, “I”这几种，分别对应1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1。如果在这些数字之间的，比如3，那就重复3次写1：III。所以只需要按照罗马数字表示的从大到小顺序，当num比a[i]大的时候，cnt = num / a[i],就将b[i]连接在result字符串后面cnt次。每一次循环将num取余a[i]，直到num<=0为止～

```
class Solution {
public:
    string intToRoman(int num) {
        string result = "";
```

```
        int a[] = {1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
        string b[] = {"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"};
        int index = 0, cnt;
        while(num > 0) {
            cnt = num / a[index];
            while(cnt--)
                result += b[index];
            num = num % a[index];
            index++;
        }
        return result;
    }
};
```

013. Roman to Integer [Easy]

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

```
class Solution {
public:
    int romanToInt(string s) {
        int ans = 0;
        map<char, int> m;
        char c[] = {'I', 'V', 'X', 'L', 'C', 'D', 'M'};
        int t[] = {1, 5, 10, 50, 100, 500, 1000};
        for(int i = 0; i < 7; i++) {
            m.insert(pair<char, int> (c[i], t[i]));
        }
        for(int i = 0; i < s.length() - 1; i++) {
            if(m[s[i]] >= m[s[i + 1]]) {
                ans = ans + m[s[i]];
            } else {
                ans = ans - m[s[i]];
            }
        }
        ans = ans + m[s[s.length() - 1]];
        return ans;
    }
};
```

014. Longest Common Prefix [Easy]

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

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

015. 3Sum [Medium]

Given an array S of n integers, are there elements a, b, c in S such that a + 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]]

题目大意：给一个整数数组，从中选三个元素abc，使a + b + c = 0，返回所有满足条件的abc集合，结果集合中的结果不能有重复～

分析：将sum数组排序，先确定nums[i]为第一个元素，为了避免重复，如果nums[i]和刚刚的nums[i-1]相同就跳过continue，然后begin指向i+1，end指向n-1，判断此时的sum是否等于0，如果等于0就将结果放入result数组中，且begin++，end – -，为了避免重复，如果begin++后的元素依旧和刚才的元素相同，继续begin++，end同理～如果sum>0就将end – -，如果sum<0就将begin++，最后返回result结果集～～

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> result;
        int n = nums.size();
        if(n < 3) return result;
        sort(nums.begin(), nums.end());
        vector<int> temp(3);
        for(int i = 0; i < n; i++) {
```

```
            if(nums[i] > 0) break;
            if(i > 0 && nums[i] == nums[i-1]) continue;
            int begin = i + 1, end = n - 1;
            while(begin < end) {
                int sum = nums[i] + nums[begin] + nums[end];
                if(sum == 0) {
                    temp[0] = nums[i];
                    temp[1] = nums[begin];
                    temp[2] = nums[end];
                    result.push_back(temp);
                    begin++;
                    end--;
                    while(begin < end && nums[begin] == nums[begin - 1]) begin++;
                    while(begin < end && nums[end] == nums[end + 1]) end--;
                } else if(sum > 0) {
                    end--;
                } else {
                    begin++;
                }
            }
        }
        return result;
    }
};
```

016. 3Sum Closest [Medium]

Given an array S of n integers, find three integers in S such 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).

题目大意：给定一个整型数组，在数组中找xyz，使x+y+z最接近target，返回最接近的x+y+z的值～

分析：对nums排序，先确定第一个数为nums[i], 使begin = i + 1，end = n – 1,如果当前sum == target，就令begin++，end–，指针分别向前向后移动一个，如果sum比较大，就令end往前一个；如果sum比较小，就令begin往后一个～每次根据sum更新result的值，result设置为long型，避免一开始是INT最大值、加了负数后溢出～

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        long result = INT_MAX;
        int n = nums.size();
```

```
        sort(nums.begin(), nums.end());
        for(int i = 0; i < n; i++) {
            int begin = i + 1, end = n - 1;
            while(begin < end) {
                int sum = nums[i] + nums[begin] + nums[end];
                if(sum == target) {
                    begin++;
                    end--;
                } else if(sum > target) {
                    end--;
                } else {
                    begin++;
                }
                if(abs(sum - target) < abs(result - target))
                    result = sum;
            }
        }
        return (int)result;
    }
};
```

017. Letter Combinations of a Phone Number [Medium]

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.

题目大意：给一个数字字符串，求可能出现的电话上的字母组合的所有情况～返回的次序可以不同～

分析：result为要返回的string数组，依次根据digits字符串的数字顺序从左到右遍历的顺序往里面添加字母，每一次都追加在原有result的后面，因为result会改变所以每次设立一个空string数组temp，然后temp根据result中原有的结果向后面继续添加拼接原+新的字符串，然后result = temp进行复制～如果digits不包含字符，应该返回空的result；否则因为要在原有result基础上拼接添加，所以先在result中push_back一个空串，以便后来的拼接字符串～

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        vector<string> result;
        if(digits.length() == 0)
            return result;
        result.push_back("");
        vector<string> v = {"", "", "abc", "def", "ghi", "jkl", "mno",
"pqrs", "tuv", "wxyz"};
        for(int i = 0; i < digits.size(); i++) {
            string s = v[digits[i] - '0'];
```

```
            vector<string> temp;
            for(int j = 0; j < s.length(); j++)
                for(int k = 0; k < result.size(); k++)
                    temp.push_back(result[k] + s[j]);
            result = temp;
        }
        return result;
    }
};
```

018. 4Sum [Medium]

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note: The solution set must not contain duplicate quadruplets.

For example, given array S = [1, 0, -1, 0, -2, 2], and target = 0.

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

题目大意：给一个整型数组，求abcd序列，使得a+b+c+d=target，返回所有不重复的abcd序列结果集合～

分析：对整型数组进行排序，先用i和j确定了前两个元素，然后用begin和end分别从j+1和最后一个元素n-1开始查找，根据sum的值移动begin和end指针，如果sum==target，就将结果放入结果集中；如果sum>target，将end向前移动一个，如果sum<target，就讲begin向后移动一个.....为了避免重复，当i、j、begin、end和它们的前一个元素相同的时候，就跳过当前元素，直接移动到下一个～

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
       vector<vector<int>> result;
       int n = nums.size();
       if(n < 4) return result;
       sort(nums.begin(), nums.end());
       vector<int> temp(4);
       for(int i = 0; i < n - 3; i++) {
            if(i != 0 && nums[i] == nums[i-1]) continue;
           for(int j = i + 1; j < n - 2; j++) {
               if(j != i + 1 && nums[j] == nums[j-1]) continue;
               int begin = j + 1, end = n - 1;
               while(begin < end) {
                   int sum = nums[i] + nums[j] + nums[begin] +
nums[end];
                   if(sum == target) {
                       temp[0] = nums[i];
                       temp[1] = nums[j];
                       temp[2] = nums[begin];
```

```
                        temp[3] = nums[end];
                        result.push_back(temp);
                        begin++;
                        end--;
                        while(begin < end && nums[begin] == nums[begin-1]) begin++;
                        while(begin < end && nums[end] == nums[end+1]) end--;
                    } else if(sum > target) {
                        end--;
                    } else {
                        begin++;
                    }
                }
            }
        }
        return result;
    }
};
```

019. Remove Nth Node From End of List [Easy]

Given a linked list, remove the nth node 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 becomes 1->2->3->5. Note: Given n will always be valid. Try to do this in one pass.

```
/**
 * 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) {
        if(head == NULL || head->next == NULL)
            return NULL;
        ListNode *p = head;
        ListNode *q = head;
        for(int i = 1; i <= n; i++) {
            p = p->next;
```

```
        }
        if(p == NULL) {
            head = head->next;
            return head;
        }
        p = p->next;
        while(p != NULL) {
            p = p->next;
            q = q->next;
        }
        q->next = q->next->next;
        return head;
    }
};
```

020. Valid Parentheses [Easy]

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> t;
        for(int i = 0; i < s.length(); i++) {
            if(s[i] == '(' || s[i] == '[' || s[i] == '{') {
                t.push(s[i]);
            } else if(s[i] == ')') {
                if(i == 0 ||  t.empty() || t.top() != '(') {
                    return false;
                }
                t.pop();
            } else if(s[i] == ']') {
                if(i == 0 || t.empty() || t.top() != '[') {
                    return false;
                }
                t.pop();
            } else if(s[i] == '}') {
                if(i == 0 || t.empty() || t.top() != '{') {
                    return false;
                }
                t.pop();
            }
        }
        return t.empty();
    }
```

```
};
```

021. Merge Two Sorted Lists [Easy]

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l1 != NULL && l2 != NULL) {
            ListNode *p = l1;
            ListNode *q = l2;
            ListNode *t, *h;//t为新链表的连接指针，h为新链表的头指针
            if(p->val > q->val) {
                t = q;
                h = q;
                q = q->next;
            } else {
                t = p;
                h = p;
                p = p->next;
            }
            while(p != NULL && q != NULL) {
                if(p->val > q->val) {
                    t->next = q;
                    t = t->next;
                    q = q->next;
                } else {
                    t->next = p;
                    t = t->next;
                    p = p->next;
                }

            }
            while(p != NULL && q == NULL) {
                t->next = p;
                p = p->next;
                t = t->next;
```

```
            }
            while(p == NULL && q != NULL) {
                t->next = q;
                q = q->next;
                t = t->next;
            }
            while(p == NULL && q == NULL) {
                return h;
            }
        }
        if(l1 == NULL && l2 != NULL) {
            return l2;
        }
        if(l1 != NULL && l2 == NULL) {
            return l1;
        }
        if(l1 == NULL && l2 == NULL) {
            return NULL;
        }
    }
};
```

022. Generate Parentheses [Medium]

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

[“((()))”, “(()())”, “(())()”, “()(())”, “()()()”]

题目大意：给n对括号，写一个方法，生成所有格式正确的括号组合～

分析：left表示还剩余的左括号个数，right表示目前需要的右括号数～一开始left = n，right = 0；当left还剩余左括号数大于0的时候，添加左括号，并将left – 1，right + 1；当right目前还需要的右括号数大于0的时候，添加右括号，并将right – 1～当left和right都等于0的时候，表示当前是一个符合格式条件的字符串，将该字符串cur放入result数组中，最后返回result～

```
class Solution {
public:
    vector<string> generateParenthesis(int n) {
        dfs("", n, 0);
        return result;
    }
private:
    vector<string> result;
    void dfs(string cur, int left, int right) {
        if (left == 0 && right == 0) {
            result.push_back(cur);
```

```
            return;
        }
        if (left > 0) dfs(cur + "(", left - 1, right + 1);
        if (right > 0) dfs(cur + ")", left, right - 1);
    }
};
```

024. Swap Nodes in Pairs [Easy]

Given a linked list, swap every two adjacent nodes and return its head.

For example, Given 1->2->3->4, you should return the list as 2->1->4->3.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

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

```
            }
        }
    }
};
```

026. Remove Duplicates from Sorted Array [Easy]

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example, Given input array nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It does not matter what you leave beyond the new length.

```
//才明白这类题目的意思是 返回的是 length 的值 但是会检验你是不是删除了相同的数字使得符合条件。。然后他检验的时候输出的只是0~length-1的值。。后面的是否符合不用管。。。
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if(nums.size() == 0) {
            return 0;
        }
        int len = 1;
        for(int i = 1; i < nums.size(); i++) {
            if(nums[i] != nums[i - 1]) {
                nums[len++] = nums[i];
            }
        }
        return len;
    }
};
```

027. Remove Element [Easy]

Given an array and a value, remove all instances of that value in place and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

The order of elements can be changed. It does not matter what you leave beyond the new length.

Example: Given input array nums = [3,2,2,3], val = 3

Your function should return length = 2, with the first two elements of nums being 2.

```cpp
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int len = 0;
        for(int i = 0; i < nums.size(); i++) {
            if(nums[i] != val) {
                nums[len++] = nums[i];
            }
        }
        return len;
    }
};
```

028. Implement strStr() [Easy]

Implement strStr().

Returns the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

```cpp
class Solution {
public:
    int strStr(string haystack, string needle) {
        int len1 = haystack.length();
        int len2 = needle.length();
        if(len2 == 0)
            return 0;
        for(int i = 0; i <= len1 - len2; i++) {
            for(int j = 0; j < len2 && needle[j] == haystack[i + j];
j++) {
                if(j == len2 - 1)
                    return i;
            }
        }
        return -1;
    }
};
```

029. Divide Two Integers [Medium]

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX_INT.

题目大意：不用乘法、除法、取余操作，将两个数相除，如果它溢出了，返回MAX_INT～

分析：我用了减法和log函数。。就是e的(loga-logb)次方等于e的log(a/b) = a/b。。。。这应该不算投机取巧吧？。。。～

```cpp
class Solution {
public:
    int divide(int dividend, int divisor) {
        if(divisor == 0 || dividend == INT_MIN && divisor == -1) return INT_MAX;
        int sign = ((dividend >> 31) ^ (divisor >> 31)) == 0 ? 1 : -1;
        long a = abs((long)dividend);
        long b = abs((long)divisor);
        double c = exp(log(a) - log(b)) + 0.0000000001;
        return (int)(sign * c);
    }
};
```

031. Next Permutation [Medium]

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column. 1,2,3 → 1,3,2 3,2,1 → 1,2,3 1,1,5 → 1,5,1

题目大意：实现下一个排列，也就是按照字典序的下一个比它大的排列～

分析：用C++库函数作个弊。。。

```cpp
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        next_permutation(nums.begin(), nums.end());
    }
};
```

033. Search in Rotated Sorted Array [Medium]

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

题目大意：假设按照升序排列的数组在事先未知的某个关键点上旋转。给一个目标数target，找这个数是否存在于这个序列中，如果存在返回下标，不存在返回-1

分析：二分搜索法，如果当前nums[mid] < nums[l]，说明mid在旋转节点的右边，那么如果target也在mid和r之间就将l = mid + 1表示在mid + 1到r的区域找，否则将r = mid – 1在l到mid – 1的区域寻找；如果当前nums[mid] > nums[r]，说明mid在旋转节点的左边，那么如果target也在l和mid之间就将r = mid – 1，在l～mid-1的区域内寻找，否则在mid+1～r的区域内寻找；否则说明当前区间的顺序是正确的，就判断target和mid的大小关系，如果比mid所指数字大则在右边找，否则在左边找

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int l = 0, r = nums.size() - 1;
        while (l <= r) {
            int mid = (l + r) / 2;
            if (nums[mid] == target) return mid;
            if (nums[mid] < nums[l]) {
                if (target > nums[mid] && target <= nums[r])
                    l = mid + 1;
                else
                    r = mid - 1;
            } else if (nums[mid] > nums[r]) {
                if (target >= nums[l] && target < nums[mid])
                    r = mid - 1;
                else
                    l = mid + 1;
            } else {
                if (target < nums[mid])
                    r = mid - 1;
                else
                    l = mid + 1;
            }
        }
        return -1;
    }
};
```

034. Search for a Range [Medium]

Given a sorted array of integers, find the starting and ending position of a given target value.

Your algorithm's runtime complexity must be in the order of O(log n).

If the target is not found in the array, return [-1, -1].

For example, Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].

```cpp
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        int low = 0, high = nums.size() - 1;
        while(low <= high && nums[low] != target && nums[high] != target) {
            int mid = (high - low) / 2 + low;
            if(nums[mid] < target) {
                low = mid + 1;
            } else if(nums[mid] > target){
                high = mid - 1;
            } else {
                break;
            }
        }
        vector<int> v(2);
        if(low > high) {
            v[0] = -1;
            v[1] = -1;
            return v;
        }
        while(nums[low] != target) {
            low++;
        }
        while(nums[high] != target) {
            high--;
        }
        v[0] = low;
        v[1] = high;
        return v;
    }
};
```

035. Search Insert Position [Medium]

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples. [1,3,5,6], 5 → 2 [1,3,5,6], 2 → 1 [1,3,5,6], 7 → 4 [1,3,5,6], 0 → 0

```cpp
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        for(int i = 0; i < nums.size(); i++) {
            if(nums[i] >= target) {
                return i;
```

```
            }
            if(i == nums.size() - 1) {
                return nums.size();
            }
        }
        return 0;
    }
};
```

036. Valid Sudoku [Easy]

Determine if a Sudoku is valid, according to: Sudoku Puzzles – The Rules.

The Sudoku board could be partially filled, where empty cells are filled with the character '.'. A partially filled sudoku which is valid.

Note: A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated.

```
class Solution {
public:
    bool isValidSudoku(vector<vector<char>>& board) {
        int book[10];
        memset(book, 0, sizeof(int)*10);
        //每一列
        for(int i = 0; i < 9; i++) {
            for(int j = 0; j < 9; j++) {
                if(board[i][j] == '.') continue;
                if(book[board[i][j] - '0'] == 0)
                    book[board[i][j] - '0'] = 1;
                else
                    return false;
            }
            memset(book, 0, sizeof(int) * 10);
        }
        //每一行
        for(int i = 0; i < 9; i++) {
            for(int j = 0; j < 9; j++) {
                if(board[j][i] == '.') continue;
                if(book[board[j][i] - '0'] == 0)
                    book[board[j][i] - '0'] = 1;
                else
                    return false;
            }
            memset(book, 0, sizeof(int) * 10);
        }
        //每个小九宫格
        for(int m = 0; m <= 6; m = m + 3) {
```

```
            for(int i = 0; i <= 8; i++) {
                if(i % 3 == 0)
                     memset(book, 0, sizeof(int) * 10);
                for(int j = 0; j <= 2; j++) {
                    if(board[i][j + m] == '.') continue;
                    if(book[board[i][j + m] - '0'] == 0)
                        book[board[i][j + m] - '0'] = 1;
                    else
                        return false;
                }
            }
        }
        return true;
    }
};
```

038. Count and Say [Easy]

The count-and-say sequence is the sequence of integers beginning as follows: 1, 11, 21, 1211, 111221, ...

1 is read off as “one 1” or 11. 11 is read off as “two 1s” or 21. 21 is read off as “one 2, then one 1” or 1211. Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string.

```
class Solution {
public:
    string countAndSay(int n) {
        string temp = "";
        string num = "";
        int cnt = 1;
        if(n == 1) {
            return "1";
        }
        if(n == 2) {
            return "11";
        }
        string a = "11";
        for(int i = 1; i <= n - 2; i++) {
            for(int j = 1; j < a.length(); j++) {
                if(a[j-1] == a[j]) {
                    cnt++;
                }
                if(a[j-1] != a[j]) {
                    while(cnt) {
                        num = (char)(cnt % 10 + '0') + num;
                        cnt = cnt / 10;
```

```
                    }
                    temp += num;
                    num = "";
                    temp += a[j-1];
                    cnt = 1;
                }
                if(j == a.length() - 1) {
                    while(cnt) {
                        num = (char)(cnt % 10 + '0') + num;
                        cnt = cnt / 10;
                    }
                    temp += num;
                    num = "";
                    temp += a[j];
                    cnt = 1;
                }
            }
            a = temp;
            temp = "";
        }
        return a;
    }
};
```

039. Combination Sum [Medium]

Given a set of candidate numbers (C) (without duplicates) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note: All numbers (including target) will be positive integers. The solution set must not contain duplicate combinations. For example, given candidate set [2, 3, 6, 7] and target 7, A solution set is: [[7], [2, 2, 3]]

题目大意：给一个可选数的集合，和一个目标数target，找所有组合方式，使组合的数总和为target～集合中的数可以重复选择～

分析：深度优先搜索，每次i从index到len分别将i放入row中，如果index超过了nums的长度就return，如果sum == target就将当前结果放入result数组中，为了避免无底洞，如果重复加自己的时候（i == index的时候）考虑如果nums[i]是正数且sum 大于 target就别继续加了,直接return终止了这条路径～nums[i]是负数且sum < target的时候同理～最后返回result～

```
class Solution {
public:
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        nums = candidates;
        this->target = target;
```

```
        dfs(0, 0);
        return result;
    }
private:
    vector<int> nums;
    int target;
    vector<vector<int>> result;
    vector<int> row;
    void dfs(int index, int sum) {
        if (index > nums.size() - 1) return;
        if (sum == target) result.push_back(row);
        for (int i = index; i < nums.size(); i++) {
            if (i == index && (nums[i] > 0 && sum > target || nums[i] < 0 && sum < target)) return;
            row.push_back(nums[i]);
            dfs(i, sum + nums[i]);
            row.pop_back();
        }
    }
};
```

040. Combination Sum II [Medium]

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note: All numbers (including target) will be positive integers. The solution set must not contain duplicate combinations. For example, given candidate set [10, 1, 2, 7, 6, 1, 5] and target 8, A solution set is: [[1, 7], [1, 2, 5], [2, 6], [1, 1, 6]]

题目大意：给一个集合的数字，和一个目标数字target，找一种组合方式使组合中所有数字之和等于target，集合中每个数字只能使用一次，集合中每个数字都是正数～

分析：先对所有数字排序，之后按照深度优先搜索将index + 1 ～ len之间所有的数字都尝试放在结果集中，比较sum与target的大小，如果和target一样大，就把当前组合结果放入集合中（为了避免重复），如果比target大，因为所有数都是正数，所以要提前return（不这样做会超时的～）最后把集合中的所有结果放入一个二维数组result中，返回result～

```
class Solution {
public:
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        sort(candidates.begin(), candidates.end());
        nums = candidates;
        this->target = target;
        dfs(-1, 0);
        vector<vector<int>> result;
```

```
        for (auto it = s.begin(); it != s.end(); it++)
            result.push_back(*it);
        return result;
    }
private:
    vector<int> nums;
    int target;
    set<vector<int>> s;
    vector<int> row;
    void dfs(int index, int sum) {
        if (sum == target) {
            s.insert(row);
            return;
        } else if (sum > target) {
            return;
        }
        for (int i = index + 1; i < nums.size(); i++) {
            row.push_back(nums[i]);
            dfs(i, sum + nums[i]);
            row.pop_back();
        }
    }
};
```

042. Trapping Rain Water [Hard]

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example: Given num = 16, return true. Given num = 5, return false.

Follow up: Could you solve it without loops/recursion?

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

```
class Solution {
public:
    bool isPowerOfFour(int num) {
        return num <= 0 ? false : pow(4, (round)(log(num) / log(4))) ==
num;
    }
};
```

043. Multiply Strings [Medium]

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2.

Note:

The length of both num1 and num2 is < 110. Both num1 and num2 contains only digits 0-9. Both num1 and num2 does not contain any leading zero. You must not use any built-in BigInteger library or convert the inputs to integer directly.

题目大意：给两个用字符串表示的非负整数，计算他们的乘积，结果用string表示～

分析：先将字符串num1和num2倒置，然后i和j分别遍历num1和num2，将num1[i] * num2[j]的结果加上v[i+j]本身的结果保存为temp，temp的余数保存在v[i+j]中，temp的进位累加在v[i+j+1]中。从后向前从第一个非0数字开始将数字转化为字符保存在result字符串中，result即为所求字符串～

```
class Solution {
public:
    string multiply(string num1, string num2) {
        string result = "";
        reverse(num1.begin(), num1.end());
        reverse(num2.begin(), num2.end());
        int len1 = num1.length(), len2 = num2.length();
        vector<int> v(len1 + len2, 0);
        for (int i = 0; i < len1; i++) {
            for (int j = 0; j < len2; j++) {
                int temp = v[i + j] + (num1[i] - '0') * (num2[j] - '0');
                v[i + j] = temp % 10;
                v[i + j + 1] += temp / 10;
            }
        }
        int flag = 0;
        for (int i = len1 + len2 - 1; i >= 0; i--) {
            if (flag == 0 && v[i] != 0) flag = 1;
            if (flag == 1) result += to_string(v[i]);
        }
        return result == "" ? "0" : result;
    }
};
```

046. Permutations [Medium]

Given a collection of distinct numbers, return all possible permutations.

For example, [1,2,3] have the following permutations: [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

题目大意：给一个不同数字的集合，返回它的所有可能的排列～

分析：对nums进行排序，然后将所有全排列结果放入result数组中返回～

```cpp
class Solution {
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> result;
        sort(nums.begin(), nums.end());
        do {
            result.push_back(nums);
        } while (next_permutation(nums.begin(), nums.end()));
        return result;
    }
};
```

047. Permutations II [Medium]

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

For example, [1,1,2] have the following unique permutations: [[1,1,2], [1,2,1], [2,1,1]]

题目大意：返回一个数字集合构成的数字（可能有重复数字）的所有可能的唯一的全排列～

分析：先对nums排序，然后将所有全排列加入result中～

```cpp
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        vector<vector<int>> result;
        sort(nums.begin(), nums.end());
        do {
            result.push_back(nums);
        } while (next_permutation(nums.begin(), nums.end()));
        return result;
    }
};
```

048. Rotate Image [Medium]

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this in-place?

```cpp
//原地旋转90度。
//1.沿主对角线所有元素交换
//2.沿着垂直中轴线方向所有元素交换

```

```
class Solution {
public:
    void rotate(vector<vector<int>>& matrix) {
        for(int i = 0; i < matrix.size(); i++)
            for(int j = 0; j <= i; j++)
                swap(matrix[i][j], matrix[j][i]);

        for(int i = 0, j = matrix.size() - 1; i < j; i++, j--)
            for(int k = 0; k < matrix.size(); k++)
                swap(matrix[k][i], matrix[k][j]);
    }
};
```

049. Group Anagrams [Medium]

Given an array of strings, group anagrams together.

For example, given: [“eat”, “tea”, “tan”, “ate”, “nat”, “bat”], Return:

[[“ate”, “eat”,”tea”], [“nat”,”tan”], [“bat”]] Note: All inputs will be in lower-case.

题目大意：给一组字符串，将这些字符串分组，按照同字母异序词为一组

分析：每一个s排序后的字符串为t，将s保存在以t为键的map中，这样同字母异序词即为一组。遍历map中的每一个vector<string>，将其保存在ans中，ans即为结果～

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        vector<vector<string>> ans;
        map<string, vector<string>> mp;
        for (auto s : strs) {
            string t = s;
            sort(t.begin(), t.end());
            mp[t].push_back(s);
        }
        for (auto m : mp) {
            vector<string> temp(m.second.begin(), m.second.end());
            ans.push_back(temp);
        }
        return ans;
    }
};
```

051. N-Queens [Hard]

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other. Given an integer n, return all distinct solutions to the n-queens puzzle. Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively. For example, There exist two distinct solutions to the 4-queens puzzle: [[".Q..", // Solution 1 "...Q", "Q...", "..Q."],

["..Q.", // Solution 2 "Q...", "...Q", ".Q.."]]

分析：pos[i]存放的是第i行的皇后所在的位置，递归以行的形式递归，每次放置的皇后要判断是否与前面已经放置的皇后冲突。从pos[row] = 0开始一直到n-1，判断是否安全 如果安全就进行下一行的摆放，每次递归到row==n的时候表示当前所有n个皇后已经摆放完成，此时将当前完成的结果保存在string类型的temp数组里面，先将数组置为'.....'，后根据pos[i]存放i行皇后的位置的特性将temp数组里面temp[i][pos[i]]置为'Q'。然后将temp压入v数组中，return。这样递归结束就能找到所有的摆放方法。这是一个深度优先的过程，从在第一行放在第一个位置开始，摆放第二行、第三行...直到最后一行。然后pos[row]++,表示将第一行放在第二个位置...然后摆放第二行、第三行...直到最后一行..... 直到所有的情况深度优先搜索完成～

```
class Solution {
    vector<vector<string> > v;
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<int> pos(n);
        dfs(pos, n, 0);
        return v;
    }
private:
    void dfs(vector<int> &pos, int n, int row) {
        if(row == n) {
            vector<string> temp(n, string(n, '.'));
            for(int i = 0; i < n; i++) {
                temp[i][pos[i]] = 'Q';
            }
            v.push_back(temp);
            return ;
        }
        for(pos[row] = 0; pos[row] < n; pos[row]++) {
            if(issafe(pos, n, row)) {
                dfs(pos, n, row + 1);
            }
        }
    }

    bool issafe(vector<int> &pos, int n, int row) {
        for(int i = 0; i < row; i++)
            if(pos[i] == pos[row] || abs(i - row) == abs(pos[i] -
pos[row]))
                return false;
        return true;
```

```
    }
};
```

052. N-Queens II [Hard]

Follow up for N-Queens problem. Now, instead outputting board configurations, return the total number of distinct solutions.

分析：和LeetCode 51. N-Queens一样，只需改动几行代码即可~

```
class Solution {
    int cnt = 0;
public:
    int totalNQueens(int n) {
        vector<int> pos(n);
        dfs(pos, n, 0);
        return cnt;
    }
private:
    void dfs(vector<int> &pos, int n, int row) {
        if(row == n) {
            cnt++;
            return ;
        }
        for(pos[row] = 0; pos[row] < n; pos[row]++) {
            if(issafe(pos, n, row)) {
                dfs(pos, n, row + 1);
            }
        }
    }

    bool issafe(vector<int> &pos, int n, int row) {
        for(int i = 0; i < row; i++)
            if(pos[i] == pos[row] || abs(i - row) == abs(pos[i] - pos[row]))
                return false;
        return true;
    }
};
```

053. Maximum Subarray [Medium]

Find the contiguous subarray within an array (containing at least one number) which has the largest sum. For example, given the array [−2,1,−3,4,−1,2,1,−5,4], the contiguous subarray [4,−1,2,1] has the largest sum = 6. click to show more practice. More practice: If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.

分析：特例是，当所有数都为负数的时候，要返回一个最小的负数，而非返回0。设temp的初始化值为nums[0]，i从1一直遍历到len-1：

0.ans始终为temp和ans值中较大的那一个。

1.当当前temp的值为正数的时候，来者nums[i]加temp。//此时如果nums[i]为负数对临时总和temp无贡献，则不会更新ans的值，我们临时把它收入temp的总和当中以备后用。如果nums[i]是正数，对临时总和temp有贡献，那就会更新ans的最大值。

2.当当前temp的值为负数的时候，temp的值直接=nums[i]。//之前的临时总和temp是个负数，对于来者nums[i]来说不管怎样nums[i]如果+temp都是在减少nums[i]，还不如直接将temp=0舍弃前面的负数和，取nums[i]当作当前的临时总和的值。

3.temp如果为0不用考虑怎样都行~

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

055. Jump Game [Medium]

Given an array of non-negative integers, you are initially positioned at the first index of the array. Each element in the array represents your maximum jump length at that position. Determine if you are able to reach the last index. For example: A = [2,3,1,1,4], return true. A = [3,2,1,0,4], return false.

题目解释：从下标为0的地方开始，A[i]表示当前i处能够跳跃的最大长度。（也就是也就是i处最远能跳到下标i + nums[i]处。）判断能不能跳啊跳到最后一个下标的地方。

分析：设立distance为当处在i下标的时候，前面所能够达到的所有长度的最大值（因为是最大值，所以0~最大值的所有下标都可以遍历到），当i <= distance的所有下标都可以遍历，然后更新distance的值为distance = max(distance, i + nums[i]); 最后判断能够最远到达的distance是否够的到最后一个下标n-1，不能的话返回false，能的话返回true

```cpp
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int distance = 0;
        for(int i = 0; i < nums.size() - 1 && i <= distance; i++) {
            distance = max(distance, i + nums[i]);
        }
        return distance >= (nums.size() - 1);
    }
};
```

056. Merge Intervals [Medium]

Given a collection of intervals, merge all overlapping intervals.

For example, Given [1,3],[2,6],[8,10],[15,18], return [1,6],[8,10],[15,18].

题目大意：给一组区间，合并他们所有的重叠区间

分析：先对所有区间按左区间从小到大排序，排序后将第一个区间放入ans结果数组中，遍历原数组，如果当前ans数组的最后一个区间的end小于intervals[i]的start，说明与其没有交集，那么就将这个intervals[i]放入结果数组中。否则说明有交集，就将当前ans数组的最后一个区间的end更新为ans.back().end和intervals[i].end()中较大的那一个，最后返回ans数组即为所求～注意如果intervals中不含元素要返回空集～

```cpp
/**
 * Definition for an interval.
 * struct Interval {
 *     int start;
 *     int end;
 *     Interval() : start(0), end(0) {}
 *     Interval(int s, int e) : start(s), end(e) {}
 * };
 */
class Solution {
public:
    vector<Interval> merge(vector<Interval>& intervals) {
        vector<Interval> ans;
        if (intervals.size() == 0) return ans;
        sort(intervals.begin(), intervals.end(), [](Interval a,
Interval b){return a.start < b.start;});
        ans.push_back(intervals[0]);
        for (int i = 1; i < intervals.size(); i++) {
            if(ans.back().end < intervals[i].start)
                ans.push_back(intervals[i]);
            else
```

```
                ans.back().end = max(ans.back().end, intervals[i].end);
        }
        return ans;
    }
};
```

058. Length of Last Word [Easy]

Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example, Given s = "Hello World", return 5.

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int cnt = 0;
        int flag = 0;
        for(int i = s.length() - 1; i >= 0; i--) {
            if(flag == 0 && s[i] == ' ') {
                continue;
            }
            if(s[i] != ' ') {
                flag = 1;
                cnt++;
            } else {
                break;
            }
        }
        return cnt;
    }
};
```

059. Spiral Matrix II [Medium]

Given an integer n, generate a square matrix filled with elements from 1 to n2 in spiral order.

For example, Given n = 3,

You should return the following matrix: [[1, 2, 3], [8, 9, 4], [7, 6, 5]]

题目大意：给一个n，输出一个n*n的数组，并且按照螺旋的方式填充入数字1～n*n。

分析：按照一个个矩阵的边框输入：x为矩阵的上界，n为矩阵的上界，每次输出这个围成的矩阵的第一行——最后一列——最后一行——第一列，然后将x自增1，m自减1～ 注意：为了避免重复输出，当x和n相等的时候，就输入一次第一行和最后一列就可以，不用重复输入最后一行和第一列～

```cpp
class Solution {
public:
    vector<vector<int>> generateMatrix(int n) {
        vector<vector<int>> result(n, vector<int>(n));
        n = n - 1;
        int num = 1;
        for (int x = 0; x <= n; x++, n--) {
            for (int j = x; j <= n; j++)
                result[x][j] = num++;
            for (int i = x + 1; i <= n - 1; i++)
                result[i][n] = num++;
            for (int j = n; j >= x && x != n; j--)
                result[n][j] = num++;
            for (int i = n - 1; i >= x + 1 && x != n; i--)
                result[i][x] = num++;
        }
        return result;
    }
};
```

060. Permutation Sequence [Medium]

The set [1,2,3,...,n] contains a total of n! unique permutations. By listing and labeling all of the permutations in order, We get the following sequence (ie, for n = 3): “123” “132” “213” “231” “312” “321” Given n and k, return the kth permutation sequence.

题目大意：返回1, 2, 3...n的第k个全排列～

分析：result一开始为1 2 3 4 ... n，用C++库函数，当到第k个全排列的时候返回result～

```cpp
class Solution {
public:
    string getPermutation(int n, int k) {
        string result = "";
        for (int i = 1; i <= n; i++)
            result += to_string(i);
        do {
            k--;
        } while (k > 0 && next_permutation(result.begin(),
result.end()));
        return result;
    }
};
```

061. Rotate List [Medium]

Given a list, rotate the list to the right by k places, where k is non-negative.

For example: Given 1->2->3->4->5->NULL and k = 2, return 4->5->1->2->3->NULL.

题目大意：将一个链表向右循环k次，返回这个链表～

分析：计算出整个链表的长度len，如果要向右循环k次，则新的head指针应该在往右移动len – k % len处。（如果向右移动的距离moveDistance == len，那么直接返回head即可），newhead之前的一个指针的next应为NULL。并且尾部NULL前的tail指针处，tail的next应该为原来的head，最后返回newhead～

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (head == NULL) return head;
        int len = 0;
        ListNode *newhead = head, *tail = head, *p = head;
        while (p != NULL) {
            if (p->next == NULL)
                tail = p;
            len++;
            p = p->next;
        }
        int moveDistance = len - k % len;
        if (moveDistance == len) return head;
        for (int i = 0; i < moveDistance - 1; i++) {
            newhead = newhead->next;
        }
        ListNode *temp = newhead;
        newhead = newhead->next;
        temp->next = NULL;
        tail->next = head;
        return newhead;
    }
};
```

062. Unique Paths [Medium]

A robot is located at the top-left corner of a m x n grid (marked ‘Start’ in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there? Above is a 3 x 7 grid. How many possible unique paths are there?

Note: m and n will be at most 100.

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        int a[100][100];
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(i == 0 || j == 0)
                    a[i][j] = 1;
                else
                    a[i][j] = a[i - 1][j] + a[i][j - 1];
            }
        }
        return a[m - 1][n - 1];
    }
};
```

063. Unique Paths II [Medium]

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

For example, There is one obstacle in the middle of a 3×3 grid as illustrated below.

[[0,0,0], [0,1,0], [0,0,0]] The total number of unique paths is 2.

Note: m and n will be at most 100.

分析：这道题是上一道题的升级版~加了障碍物~加不加障碍物差别就在于，当当前地域有障碍物的时候，a[i][j] = 0，其余的不变： 0.a[0][0] = 1; 1.对于i==0的时候，为最上面一排，当前方格只能由左边方格来，所以a[i][j] = a[i][j-1]; 2.对于j==0的时候，为最左边一排，当前方格只能由上边方格来，所以a[i][j] = a[i-1][j]; 3.其余情况，当前方格能由左边和上边两个方向过来，所以a[i][j] = a[i-1][j] + a[i][j-1]; 最后直到一直递推输出到终点(m-1, n-1)的时候return a[m-1][n-1];

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
```

```
        int m = obstacleGrid.size();
        int n = obstacleGrid[0].size();
        int a[100][100];
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(obstacleGrid[i][j] == 1) {
                    a[i][j] = 0;
                } else if(i == 0 && j == 0) {
                    a[i][j] = 1;
                } else if(i == 0) {
                    a[i][j] = a[i][j-1];
                } else if(j == 0) {
                    a[i][j] = a[i-1][j];
                } else {
                    a[i][j] = a[i-1][j] + a[i][j-1];
                }
            }
        }
        return a[m-1][n-1];
    }
};
```

064. Minimum Path Sum [Medium]

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

分析：用一个dp二维数组标记当前方格的最小值～

0.当i == 0 && j == 0的时候，dp[i][j] = grid[i][j];

1.对于i==0的时候，为最上面一排，当前方格只能由左边方格来，所以dp[i][j] = dp[i][j-1] + grid[i][j];

2.对于j==0的时候，为最左边一排，当前方格只能由上边方格来，所以dp[i][j] = dp[i-1][j] + grid[i][j];

3.其他情况下，dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j];

最后直到一直递推输出到终点(m-1, n-1)的时候return dp[m-1][n-1];～～

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();
        if(m == 0 || n == 0)
            return 0;
        vector<vector<int>> dp(m, vector<int>(n));
        for(int i = 0; i < m; i++) {
```

```
            for(int j = 0; j < n; j++) {
                if(i == 0 && j == 0) {
                    dp[i][j] = grid[i][j];
                } else if(i == 0) {
                    dp[i][j] = dp[i][j-1] + grid[i][j];
                } else if(j == 0) {
                    dp[i][j] = dp[i-1][j] + grid[i][j];
                } else {
                    dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i]
[j];
                }
            }
        }
        return dp[m-1][n-1];
    }
};
```

066. Plus One [Easy]

Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
            for(int i = digits.size() - 1; i >= 0; i--) {
                if(digits[i] != 9) {
                    digits[i] = digits[i] + 1;
                    return digits;
                } else {
                    digits[i] = 0;
                }
            }
            digits.insert(digits.begin(), 1);

            return digits;

    }
};
```

067. Add Binary [Easy]

Given two binary strings, return their sum (also a binary string).

For example, a = “11” b = “1” Return “100”.

```cpp
class Solution {
public:
    string addBinary(string a, string b) {
        string s;
        int lena = a.length() - 1;
        int lenb = b.length() - 1;
        int temp = 0;
        char c;
        while(lena >= 0 && lenb >= 0) {
            c = (a[lena] - '0') + (b[lenb] - '0') + temp + '0';
            temp = 0;
            if((c - '0') >= 2) {
                temp = 1;
                c = c - 2;
            }
            s = c + s;
            lena--;
            lenb--;
        }
        while(lena >= 0) {
            c = (a[lena] - '0') + temp + '0';
            temp = 0;
            if((c - '0') >= 2) {
                temp = 1;
                c = c - 2;
            }
            s = c + s;
            lena--;
        }
        while(lenb >= 0) {
            c =(b[lenb] - '0') + temp + '0';
            temp = 0;
            if((c - '0') >= 2) {
                temp = 1;
                c = c - 2;
            }
            s = c + s;
            lenb--;
        }
        if(temp == 1) {
            s = '1' + s;
        }
        return s;
    }
};
```

069. Sqrt(x) [Medium]

Implement int sqrt(int x).

Compute and return the square root of x.

题目大意：实现sqrt函数，返回x的sqrt结果（int型）

分析：二分法，令left = 0， right为int最大值，mid为left和right的中间值。始终保持要求的值在left和right之间。进入循环——如果mid的平方小于等于x并且mid+1的平方大于等于x则返回mid～否则：如果mid的平方小于x，说明答案在mid的右边，left = mid + 1，mid的平方大于x，说明答案在mid的左边，right = mid – 1～ 注意：令left、right、mid都是long型因为为了避免mid*mid的结果超出整型范围～

```
class Solution {
public:
    int mySqrt(int x) {
        long left = 0, right = INT_MAX, mid = 0;
        while (true) {
            long mid = left + (right - left) / 2;
            if (mid * mid <= x && (mid + 1) * (mid + 1) > x)
                return (int)mid;
            if (mid * mid < x)
                left = mid + 1;
            else
                right = mid - 1;
        }
    }
};
```

070. Climbing Stairs [Easy]

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

```
class Solution {
public:
    int climbStairs(int n) {
        int *a = new int [n + 1];
        a[0] = 1;
        a[1] = 1;
        for(int i = 2; i <= n; i++) {
            a[i] = a[i - 1] + a[i - 2];
        }
        return a[n];
    }
};

```

```
14
15  //一开始我写了个Fibonacci递归，在n = 44 时候超时了
16  class Solution {
17  public:
18      int climbStairs(int n) {
19          if(n == 1 || n == 2)
20              return n;
21          return climbStairs(n - 1) + climbStairs(n - 2);
22      }
23  };
24
25
26  //最最一开始我用的最原始的递归， 在 n=38的时候超时了
27  class Solution {
28  public:
29      int ans = 0;
30      int climbStairs(int n) {
31          if(n == 0) {
32              ans++;
33              return ans;
34          }
35          if(n >= 1) {
36              climbStairs(n - 1);
37          }
38          if(n >= 2) {
39              climbStairs(n - 2);
40          }
41      }
42  };
```

071. Simplify Path [Medium]

Given an absolute path for a file (Unix-style), simplify it.

For example, path = “/home/”, => “/home” path = “/a/./b/../../c/”, => “/c” click to show corner cases.

Corner Cases: Did you consider the case where path = “/../”? In this case, you should return “/”. Another corner case is the path might contain multiple slashes ‘/’ together, such as “/home//foo/”. In this case, you should ignore redundant slashes and return “/home/foo”.

题目大意：简化一个Unix风格的绝对路径，返回简化后的结果～

分析：以”/”为分隔，将所有不是”.”和”..”的字符串放入栈中。如果是”..”并且上一层不为空，就返回上一层，也就是弹栈；如果是”.”，不处理；如果有多个连续的”/”，则只认一个”/”：从头到尾遍历字符串，如果是”/”就不断跳过；当不是”/”的时候，将后面的字符串放入temp中，如果是”..”就弹栈，如果不是”..”和”.”就把temp压入栈中。最后将栈中所有的元素按照”/”分隔连接成result字符串～

```
1  class Solution {
```

```
public:
    string simplifyPath(string path) {
        stack<string> s;
        string result = "", temp = "";
        int i = 0, len = path.length();
        while (i < len) {
            while (i < len && path[i] == '/') i++;
            temp = "";
            while (i < len && path[i] != '/') temp += path[i++];
            if (temp == ".." && !s.empty())
                s.pop();
            else if (temp != "" && temp != "." && temp != "..")
                s.push(temp);
        }
        if (s.empty()) return "/";
        while (!s.empty()) {
            result = "/" + s.top() + result;
            s.pop();
        }
        return result;
    }
};
```

074. Search a 2D Matrix [Medium]

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted from left to right. The first integer of each row is greater than the last integer of the previous row. For example,

Consider the following matrix:

[[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 50]] Given target = 3, return true.

题目大意：编写一个有效的算法，在m×n矩阵中搜索一个值。 该矩阵具有以下属性： 每行中的整数从左到右排序。每行的第一个整数大于前一行的最后一个整数。

分析：对每一行进行二分搜索，如果找到了就返回true否则返回false～

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        for(int i = 0; i < matrix.size(); i++)
            if (binary_search(matrix[i].begin(), matrix[i].end(),
target)) return true;
        return false;
    }
};
```

075. Sort Colors [Medium]

Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int cnt0 = 0, cnt1 = 0, cnt2 = 0;
        for(int i = 0; i < nums.size(); i++) {
            if(nums[i] == 0) {
                cnt0++;
            } else if (nums[i] == 1) {
                cnt1++;
            } else {
                cnt2++;
            }
        }
        for(int i = 0; i < cnt0; i++) {
            nums[i] = 0;
        }
        for(int i = cnt0; i < cnt0 + cnt1; i++) {
            nums[i] = 1;
        }
        for(int i = cnt1 + cnt0; i < nums.size(); i++) {
            nums[i] = 2;
        }
    }
};
```

077. Combinations [Medium]

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

For example, If n = 4 and k = 2, a solution is:

[[2,4], [3,4], [2,3], [1,2], [1,3], [1,4],]

题目大意：给两个整数n和k，返回所有的k个数字组合，这些数字只能从1...n中选择～

分析：从cur == 0，cnt == 0开始，每次将cur + 1 ～ n之间的数字放入row中，并将cnt + 1，然后继续深度优先搜索直到cnt == k为止将row放入result中作为结果之一，不要忘记dfs遍历后还要将当前元素pop_back()出来，最后返回result～

```cpp
class Solution {
public:
    vector<vector<int>> combine(int n, int k) {
        this->n = n, this->k = k;
        dfs(0, 0);
        return result;
    }
private:
    int n, k;
    vector<vector<int>> result;
    vector<int> row;
    void dfs(int cur, int cnt) {
        if (cnt == k) {
            result.push_back(row);
            return;
        }
        for (int i = cur + 1; i <= n; i++) {
            row.push_back(i);
            dfs(i, cnt + 1);
            row.pop_back();
        }
    }
};
```

078. Subsets [Medium]

Given a set of distinct integers, nums, return all possible subsets.

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], []]

题目大意：给一个集合nums，求nums的所有子集集合～

分析：用位运算，j从0到maxn变化，每一次计算j移动i位后最后一位是否为1，如果为1就将nums[i]的值放入result[j]～

```cpp
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int len = nums.size();
        int maxn = pow(2, len);
        vector<vector<int>> result(maxn);
        for (int i = 0; i < len; i++) {
            for (int j = 0; j < maxn; j++) {
                if ((j >> i) & 1)
                    result[j].push_back(nums[i]);
```

```
            }
        }
        return result;
    }
};
```

080. Remove Duplicates from Sorted Array II [Medium]

Follow up for “Remove Duplicates”: What if duplicates are allowed at most twice?

For example, Given sorted array nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of nums being 1, 1, 2, 2 and 3. It does not matter what you leave beyond the new length.

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

082. Remove Duplicates from Sorted List II [Medium]

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers 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.

题目大意：给一个已经排序的链表，删除所有元素值出现过2次或者以上的结点，只保留元素值仅仅出现过一次的结点～

分析：p为head的next，如果p的值和head的值不同，则将head->next置为deleteDuplicates(p)，返回head，否则直到找到第一个p不等于head的地方，返回deleteDuplicates(p)～

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (head == NULL || head->next == NULL) return head;
        ListNode* p = head->next;
        if (p->val != head->val) {
            head->next = deleteDuplicates(p);
            return head;
        } else {
            while (p != NULL && p->val == head->val) p = p->next;
            return deleteDuplicates(p);
        }
    }
};
```

083. Remove Duplicates from Sorted List [Easy]

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example, Given 1->1->2, return 1->2. Given 1->1->2->3->3, return 1->2->3.

```
/**
 * 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 || head->next == NULL) {
            return head;
        }
        ListNode *p = head;
        while(p != NULL &&  p->next != NULL) {
            while (p->next != NULL && p->next->val == p->val) {
                    p->next = p->next->next;
            }
```

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

088. Merge Sorted Array [Easy]

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note: You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int index = m + n - 1;
        int aindex = m - 1;
        int bindex = n - 1;
        while(aindex >= 0 && bindex >= 0) {
            if(nums1[aindex] > nums2[bindex]) {
                nums1[index--] = nums1[aindex--];
            } else {
                nums1[index--] = nums2[bindex--];
            }
        }
        while(bindex >= 0) {
            nums1[index--] = nums2[bindex--];
        }
    }
};
```

090. Subsets II [Medium]

Given a collection of integers that might contain duplicates, nums, return all possible subsets.

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], []]

题目大意：给一个集合nums，nums中可能含有重复元素，求nums的所有子集集合～集合中的元素不能重复～

分析：首先对nums进行排序，接着用位运算，j从0到maxn变化，每一次计算j移动i位后最后一位是否为1，如果为1就将nums[i]的值放入result[j]～为了保证不重复，将result的结果放入集合s中，然后将s中的结果再放入result数组中返回～

```
class Solution {
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int len = nums.size();
        int maxn = pow(2, len);
        vector<vector<int>> result(maxn);
        for (int i = 0; i < len; i++) {
            for (int j = 0; j < maxn; j++) {
                if ((j >> i) & 1)
                    result[j].push_back(nums[i]);
            }
        }
        set<vector<int>> s;
        for (int i = 0; i < result.size(); i++)
            s.insert(result[i]);
        result.resize(0);
        for (auto it = s.begin(); it != s.end(); it++)
            result.push_back(*it);
        return result;
    }
};
```

091. Decode Ways [Medium]

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：和爬楼梯的那道题leetcode 70. Climbing Stairs有类似之处～～建立一个与String等长的dp数组～0.如果说当前s[i]不等于0,那先令dp[i] = dp[i-1] 1.如果s[i-1]与s[i]能够构成1~26里面的数字，那就把s[i-1]与s[i]构成一个整体，dp[i] += dp[i-2]; 最后返回数组的最后一个值dp[len-1]

```
class Solution {
public:
    int check(char a) {
        return a != '0';
    }
    int func(char a, char b) {
        return a == '1' || a == '2' && b <= '6';
```

```
    }
    int numDecodings(string s) {
        int len = s.length();
        vector<int> dp(len, 0);
        if(len == 0 || s[0] == '0')
            return 0;
        if(len == 1)
            return check(s[0]);
        dp[0] = 1;
        dp[1] = check(s[1]) + func(s[0], s[1]);
        for(int i = 2; i < len; i++) {
            if(check(s[i]))
                dp[i] = dp[i-1];
            if(func(s[i-1], s[i]))
                dp[i] += dp[i-2];
        }
        return dp[len-1];
    }
};
```

093. Restore IP Addresses [Medium]

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

For example: Given “25525511135”,

return [“255.255.11.135”, “255.255.111.35”]. (Order does not matter)

题目大意：给一个只包含数字的字符串，返回它可能组成的所有合法IP地址～

分析：i、j、k分别是点分隔的第一段、第二段、第三段的长度，从1到3，并且要给后面的段至少空一个数字的距离～这样就可以求出s1、s2、s3、s4，判断s1、s2、s3、s4是否满足条件，满足就将该结果放入result数组中～

```
class Solution {
public:
    vector<string> restoreIpAddresses(string s) {
        vector<string> result;
        int len = s.length();
        for (int i = 1; i <= 3 && i <= len - 3; i++) {
            for (int j = 1; j <= 3 && j <= len - i - 2; j++) {
                for (int k = 1; k <= 3 && k <= len - i - j - 1; k++) {
                    string s1 = s.substr(0, i), s2 = s.substr(i, j), s3 = s.substr(i + j, k), s4 = s.substr(i + j + k, len);
                    if (isValid(s1) && isValid(s2) && isValid(s3) && isValid(s4))
                        result.push_back(s1 + "." + s2 + "." + s3 + "." + s4);
```

```
                }
            }
        }
        return result;
    }
    bool isValid(string s) {
        return (s.length() >= 1 && s.length() <= 3 && (s[0] != '0' ||
s.length() == 1) && stoi(s) <= 255);
    }
};
```

094. Binary Tree Inorder Traversal [Medium]

Given a binary tree, return the inorder traversal of its nodes' values.

For example: Given binary tree [1,null,2,3], 1 \ 2 / 3 return [1,3,2].

分析：中序遍历，先遍历左子树，再输出中间，再遍历右子树～

```
/**
 * 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> result;
    vector<int> inorderTraversal(TreeNode* root) {
        dfs(root);
        return result;
    }
    void dfs(TreeNode* root) {
        if(root == NULL) return ;
        dfs(root->left);
        result.push_back(root->val);
        dfs(root->right);
    }
};
```

096. Unique Binary Search Trees [Medium]

Given n, how many structurally unique BST's (binary search trees) that store values 1...n? For example, Given n = 3, there are a total of 5 unique BST's. 1 3 3 2 1 \ / / / \ \ 3 2 1 1 3 2 / / \ \ 2 1 2 3

分析：二分查找树的定义是，左子树节点均小于root，右子树节点均大于root～所以可以用递推的方法，把v[i]表示i个数能够构成的二叉搜索树的个数～初始化边界值是 v[0]=1,v[1]=1,v[2]=2～当i>=3的时候，若以j为root结点，v[j-1]等于root结点左边的j-1个结点能构成的BST个数～v[i-j]等于root结点右边i-j个结点能构成的BST个数~//j+1~i的种数和0~i-j的种数一样。。所以就是v[i-j]～所以v[j-1] * v[i-j]等于以j为root结点能构成的BST种数～～j可以取1~i中的任意一个值，把这些所有计算出来的总数相加就是v[i]的值～～ 所以 for(int j = 1; j <= i; j++) v[i] += v[j-1] * v[i-j];最后返回的值是v[n]的值，表示1~n能组成的BST的个数～～

```
class Solution {
public:
    int numTrees(int n) {
        vector<int> v(n+1);
        v[0] = 1;
        for(int i = 1; i <= n; i++) {
            v[i] = 0;
            if(i <= 2) {
                v[i] = i;
            } else {
                for(int j = 1; j <= i; j++) {
                    v[i] += v[j-1] * v[i-j];
                }
            }
        }
        return v[n];
    }
};
```

098. Validate Binary Search Tree [Medium]

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node’s key. The right subtree of a node contains only nodes with keys greater than the node’s key. Both the left and right subtrees must also be binary search trees. Example 1: 2 / \ 1 3 Binary tree [2,1,3], return true. Example 2: 1 / \ 2 3 Binary tree [1,2,3], return false.

题目大意：给一个二叉树，判断这个二叉树是不是合法的二叉搜索树～

分析：既然是二叉搜索树，那么按照左根右遍历后的结果一定是增序～所以只需要中序遍历一遍，判断遍历结果的数组是不是后面数一定大于前面数就可以了～

```
class Solution {
private:
    vector<int> v;
```

```
public:
    bool isValidBST(TreeNode* root) {
        if(root == NULL ||(root->left == NULL && root->right == NULL))
return true;
        inorder(root);
        for(int i = 1; i < v.size(); i++)
            if(v[i] <= v[i-1]) return false;
        return true;
    }
    void inorder(TreeNode* root) {
        if(root == NULL) return;
        inorder(root->left);
        v.push_back(root->val);
        inorder(root->right);
    }
};
```

100. Same Tree [Easy]

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

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

```
/**
 * 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 p->val == q->val && isSameTree(p->left, q->left) &&
isSameTree(p->right, q->right);
        return p == NULL && q == NULL;
    }
};
```

101. Symmetric Tree [Easy]

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

For example, this binary tree is symmetric:

1 / \ 2 2 / \ / \ 3 4 4 3 But the following is not: 1 / \ 2 2 \ \ 3 3 Note: Bonus points if you could solve it both recursively and iteratively.

```
/**
 * 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 func(TreeNode *left, TreeNode *right) {
        if(left == NULL && right == NULL)
            return true;
        if(left != NULL && right != NULL && left->val == right->val) {
            return func(left->left, right->right) && func(left->right, right->left);
        }
        return false;
    }
    bool isSymmetric(TreeNode* root) {
        if(root == NULL)
            return true;
        return func(root->left, root->right);
    }
};
```

102. Binary Tree Level Order Traversal [Easy]

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

For example: Given binary tree {3,9,20,#,#,15,7}, 3 / \ 9 20 / \ 15 7 return its level order traversal as: [[3], [9,20], [15,7]]

```
/**
 * 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<int> row;
        vector<vector<int>> v;
        queue<TreeNode *> q;
        if(root == NULL)
            return v;
        q.push(root);
        TreeNode *temp;
        while(!q.empty()) {
            int size = q.size();
            while(size--) {
                temp = q.front();
                q.pop();
                row.push_back(temp->val);
                if(temp->left != NULL) {
                    q.push(temp->left);
                }
                if(temp->right != NULL) {
                    q.push(temp->right);
                }
            }
            v.push_back(row);
            row.clear();
        }
        return v;
    }
};
```

103. Binary Tree Zigzag Level Order Traversal [Medium]

Given a binary tree, return the zigzag level order traversal 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,#,#,15,7}, 3 / \ 9 20 / \ 15 7 return its zigzag level order traversal as: [[3], [20,9], [15,7]]

分析：和层序遍历一样的代码，只需要加几行代码就行～～因为要之字型存储这个二叉树～～所以只不过在行数为双数的时候需要对当前行进行一次所有元素的倒置～可以用stack也可以用数组头尾两两交换的方法～只需要在存入二维数组vector<vector> v之前倒置好row数组，再push_back到v里面就行～

```
/**
```

```
 * 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<int> row;
        vector<vector<int>> v;
        queue<TreeNode *> q;
        if(root == NULL)
            return v;
        q.push(root);
        TreeNode *temp;
        int lev = 0;
        while(!q.empty()) {
            int size = q.size();
            while(size--) {
                temp = q.front();
                q.pop();
                row.push_back(temp->val);
                if(temp->left != NULL) {
                    q.push(temp->left);
                }
                if(temp->right != NULL) {
                    q.push(temp->right);
                }
            }
            if(lev % 2) {
                int n = row.size();
                for(int i = 0; i < n/2; i++) {
                    swap(row[i], row[n-i-1]);
                }
            }
            v.push_back(row);
            lev++;
            row.clear();
        }
        return v;
    }
};
```

104. Maximum Depth of Binary Tree [Easy]

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.

```
/**
 * 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) {
        if(root == NULL)
            return 0;
        if(root->left == NULL && root->right == NULL)
            return 1;
        if(root->left != NULL && root->right == NULL)
            return maxDepth(root->left) + 1;
        if(root->right != NULL && root->left == NULL)
            return maxDepth(root->right) + 1;
        if(root->right != NULL && root->left != NULL) {
            int leftdepth = maxDepth(root->left);
            int rightdepth = maxDepth(root->right);
            return leftdepth > rightdepth ? leftdepth + 1 : rightdepth + 1;
        }
    }
};
```

107. Binary Tree Level Order Traversal II [Easy]

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

For example: Given binary tree {3,9,20,#,#,15,7}, 3 / \ 9 20 / \ 15 7 return its bottom-up level order traversal as: [[15,7], [9,20], [3]]

```
/**
 * 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<int> row;
        vector<vector<int>> v;
        if(root == NULL) {
            return v;
        }
        queue<TreeNode *> q;
        q.push(root);
        TreeNode *temp;
        while(!q.empty()) {
            int size = q.size();
            while(size--) {
                temp = q.front();
                q.pop();
                row.push_back(temp->val);
                if(temp->left != NULL) {
                    q.push(temp->left);
                }
                if(temp->right != NULL) {
                    q.push(temp->right);
                }
            }
            v.insert(v.begin(), row);
            row.clear();
        }

        return v;
    }
};
```

108. Convert Sorted Array to Binary Search Tree [Medium]

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

题目大意：给一个升序数组，把它转化为一个高度平衡的二叉搜索树～

分析：设立left和right，mid = (left + right) / 2，每次将数组的中点mid的值为根结点的值，中点左边为根结点的左子树，右边为根结点的右子树～递归求解～

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
```

```
        return func(nums, 0, nums.size() - 1);
    }
private:
    TreeNode* func(vector<int>& nums, int left, int right) {
        if (left > right) return NULL;
        int mid = (left + right) / 2;
        TreeNode* root = new TreeNode(nums[mid]);
        root->left = func(nums, left, mid - 1);
        root->right = func(nums, mid + 1, right);
        return root;
    }
};
```

109. Convert Sorted List to Binary Search Tree [Medium]

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

题目大意：给一个升序单链表，把它转换成一个高度平衡的二叉搜索树～

分析：递归求解，找到中点，将中点的值赋值给根结点，中点之前的链表为根结点的左子树，中点之后的链表为根结点的右子树～中点以这种方式确定：设立两个指针fast和slow，它们分别从head开始，fast走两步slow走一步，当fast走到最后一个结点的时候slow正好走到中点～

```
class Solution {
public:
    TreeNode* sortedListToBST(ListNode* head) {
        return func(head, NULL);
    }
    TreeNode* func(ListNode* head, ListNode* tail) {
        ListNode *fast = head, *slow = head;
        if (head == tail) return NULL;
        while (fast != tail && fast->next != tail) {
            fast = fast->next->next;
            slow = slow->next;
        }
        TreeNode* root = new TreeNode(slow->val);
        root->left = func(head, slow);
        root->right = func(slow->next, tail);
        return root;
    }
};
```

110. Balanced Binary Tree [Easy]

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 of every node never differ by more than 1.

```
/**
 * 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 flag = 1;
    int dfs(TreeNode* root) {
        if(root == NULL)
            return 1;
        int l, r;
        if(root->left == NULL) {
           l = 0;
        } else {
           l = dfs(root->left);
        }
        if(root->right == NULL) {
           r = 0;
        } else {
           r = dfs(root->right);
        }
        if(abs(l-r) >= 2) {
            flag = 0;
        }
        return (l > r ? l : r) + 1;
    }
    bool isBalanced(TreeNode* root) {
        dfs(root);
        return flag;
    }
};
```

111. Minimum Depth of Binary Tree [Easy]

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.

```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 minDepth(TreeNode* root) {
        if(root == NULL)
            return 0;
        if(root->left == NULL && root->right == NULL)
            return 1;
        if(root->left == NULL && root->right != NULL)
            return minDepth(root->right) + 1;
        if(root->right == NULL && root->left != NULL)
            return minDepth(root->left) + 1;
        if(root->left != NULL && root->right != NULL) {
            int leftdepth = minDepth(root->left);
            int rightdepth = minDepth(root->right);
            return leftdepth < rightdepth ? leftdepth + 1 : rightdepth
+ 1;
        }
    }
};
```

112. Path Sum [Easy]

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.

```
递归方法：
/**
 * 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 == NULL)
            return false;
        if(root->left == NULL && root->right == NULL && sum == root->val)
                return true;
        if(hasPathSum(root->left, sum - root->val) || hasPathSum(root->right, sum - root->val))
            return true;
        else
            return false;
    }
};


非递归法
/**
 * 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) {
        queue<TreeNode *> q;
        if(root == NULL)
            return false;
        q.push(root);
        TreeNode *temp;
        while(!q.empty()) {
            int size = q.size();
            while(size--) {
                temp = q.front();
                q.pop();
                if(temp->left != NULL) {
                    q.push(temp->left);
                    temp->left->val += temp->val;
                }
                if(temp->right != NULL) {
                    q.push(temp->right);
                    temp->right->val += temp->val;
                }
```

```
                if(temp->left == NULL && temp->right == NULL && (temp->val == sum || temp->val == sum)) {
                        return true;
                }
            }
        }
        return false;
    }
};
```

113. Path Sum II [Medium]

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：pathSum函数中只需dfs一下然后返回result数组即可，dfs函数中从root开始寻找到底端sum == root->val的结点，如果满足就将root->val压入path数组中，path数组压入result数组中，然后将当前结点弹出，return。不满足是最后一个结点的则不断深度优先左结点、右结点，同时处理好path数组的压入和弹出～～

```
/**
 * 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>> result;
    vector<int> path;
    vector<vector<int>> pathSum(TreeNode* root, int sum) {
        dfs(root, sum);
        return result;
    }
    void dfs(TreeNode* root, int sum) {
        if(root == NULL) return ;
        if(root->val == sum && root->left == NULL && root->right == NULL) {
            path.push_back(root->val);
            result.push_back(path);
            path.pop_back();
            return ;
```

```
        }
        path.push_back(root->val);
        dfs(root->left, sum - root->val);
        dfs(root->right, sum - root->val);
        path.pop_back();
    }
};
```

118. Pascal's Triangle [Easy]

Given numRows, generate the first numRows of Pascal's triangle.

For example, given numRows = 5, Return

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

分析：大概是考察用 vector 创建二维数组的应用～

```
class Solution {
public:
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> v(numRows);
        if(numRows == 0)
            return v;
        for(int i = 0; i < numRows; i++) {
            v[i].resize(i + 1);
        }
        v[0][0] = 1;
        if(numRows == 1)
            return v;
        v[1][0] = 1;
        v[1][1] = 1;
        for(int i = 2; i < numRows; i++) {
            v[i][0] = 1;
            v[i][i] = 1;
        }
        for(int i = 2; i < numRows; i++) {
            for(int j = 1; j < i; j++) {
                v[i][j] = v[i - 1][j - 1] + v[i - 1][j];
            }
        }
        return v;
    }
};
```

119. Pascal's Triangle II [Easy]

Given an index k, return the kth row of the Pascal's triangle.

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

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

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        vector<int> v(rowIndex + 1);
        v[0] = 1;
        for(int i = 1; i <= rowIndex; i++) {
            v[i] = (long long int)v[i - 1] *(long long int)(rowIndex - i + 1) / i;
        }
        return v;
    }
};
```

120. Triangle [Medium]

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 only O(n) extra space, where n is the total number of rows in the triangle.

```
class Solution {
public:
    int minimumTotal(vector<vector<int>>& triangle) {
        for(int i = triangle.size() - 2; i >= 0; i--) {
            for(int j = 0; j <= i; j++) {
                triangle[i][j] += min(triangle[i+1][j], triangle[i+1][j+1]);
            }
        }
        return triangle[0][0];
    }
};
```

121. Best Time to Buy and Sell Stock [Easy]

Say you have an array for which the ith element is the price of a given stock on day i.

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.

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int minvalue = 99999999;
        int ans = 0;
        for(int i = 0; i < prices.size(); i++) {
            if(minvalue > prices[i]) {
                minvalue = prices[i];
            }
            prices[i] = prices[i] - minvalue;
            if(ans < prices[i]) {
                ans = prices[i];
            }
        }
        return ans;
    }
};
```

122. Best Time to Buy and Sell Stock II [Medium]

Say you have an array for which the ith 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：贪心算法，在每一小段上升序列中最大差值累加得到结果。就是说在股票价格处于上升期的时候，在最低点买入，在最高点卖出。而且可知，每一小段的最大差值就是这段序列的最后一个点的价格减去这段序列第一个点的价格，与每一次从第一个点与第二点的差值一直累加所得结果相同：ans += prices[i] – prices[i – 1];

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

125. Valid Palindrome [Easy]

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” is not a 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 palindrome.

```
class Solution {
public:
    bool isPalindrome(string s) {
        if(s.length() == 0)
            return true;
        int i = 0, j = s.length() - 1;
        while(i < j) {
            while(i < j && !isalnum(s[i])) {
                i++;
            }
            while(i < j && !isalnum(s[j])) {
                j--;
            }
            if(i < j && tolower(s[i]) != tolower(s[j])) {
                return false;
            } else {
                i++;
                j--;
            }
        }
        return true;
    }
};
```

127. Word Ladder [Medium]

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.

题目大意：给出beginWord、endWord和一个字典，找到从beginWord到endWord的最短转换序列，转换要求是：1.每次只能改变一个字母～ 2.变换过程中的中间单词必须在字典中出现～（第一个beginWord不需要出现，最后一个endWord需要在字典中出现～）

分析：用广度优先搜索～先将beginWord放入队列中，然后将队列中的每一个单词从头到尾换26个字母一遍～如果换后的单词在字典中能找到～而且没有被访问过～（如果每次都找访问过的就死循环啦，不停的变来变去变同一个咋办～）那就将这个单词放入队列中继续变换～直到有一次发现在字典中找到单词的时候，这个单词恰好是endWord为止～ 因为要返回路径长度～所以在队列中放一个string和int组成的pair一对～这样的话用string表示单词，int表示变换到当前单词的路径～比如startWord就是1～之后每次加1～因为题目给的是vector～把他们所有单词先放到dict的set集合中查找单词会方便很多～visit标记当前单词是否被访问过～

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>&
wordList) {
        set<string> dict, visit;
        for(int i = 0; i < wordList.size(); i++)
            dict.insert(wordList[i]);
        queue<pair<string, int>> q;
        q.push(make_pair(beginWord, 1));
        while(!q.empty()) {
            pair<string, int> temp = q.front();
            q.pop();
            string word = temp.first;
            for(int i = 0; i < word.length(); i++) {
                string newword = word;
                for(int j = 0; j < 26; j++) {
                    newword[i] = 'a' + j;
                    if(dict.find(newword) != dict.end() &&
visit.find(newword) == visit.end()) {
                        if(newword == endWord)
                            return temp.second + 1;
                        visit.insert(newword);
                        q.push(make_pair(newword, temp.second + 1));
                    }
                }
            }
        }
        return 0;
    }
};
```

129. Sum Root to Leaf Numbers [Medium]

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.

题目大意：给一个二叉树，上面的数字是0～9中的一个，每一个根到叶子结点的路径都代表一个数字。比如1->2->3就是组成一个数字123。找所有根到叶子结点组成数字的总和～

分析：设总和为result～从根结点开始深度优先搜索，如果已经是叶子结点（即左孩子和右孩子都为空）的时候，就将result累加root->val；否则如果左子树不为空，左子树的val值累加root->val的10倍，这样到最后叶子节点的值就变成了根结点到叶子节点组成的数字的值～

```
class Solution {
public:
    int sumNumbers(TreeNode* root) {
        dfs(root);
        return result;
    }
private:
    int result = 0;
    void dfs(TreeNode* root) {
        if (root == NULL) return;
        if (root->left == NULL && root->right == NULL) {
            result += root->val;
            return;
        }
        if (root->left != NULL) {
            root->left->val += root->val * 10;
            dfs(root->left);
        }
        if (root->right != NULL) {
            root->right->val += root->val * 10;
            dfs(root->right);
        }
    }
};
```

130. Surrounded Regions [Medium]

Given a 2D board containing ‘X’ and ‘O’ (the letter O), capture all regions surrounded by ‘X’.

A region is captured by flipping all ‘O’s into ‘X’s in that surrounded region.

For example, X X X X X O O X X X O X X O X X After running your function, the board should be: X X X X X X X X X X X X O X X

题目大意：给一个地图，X表示围墙，找出所有被X围墙包围的O，并且把被包围的O替换成X～

分析：我的方法是与其找被包围的O，不如反过来寻找没有被包围的O～从地图的外围一圈开始寻找～如果当前位置是O～那就找他的上下左右～把与这个O联通的所有O都标记为"*"～标记为*后，所有没有被标记为*的O就是被包围的O～那就将所有剩余的O标记为X，把所有*标记为O～返回这张地图就可以了～

```
class Solution {
private:
    int m, n;
public:
    void solve(vector<vector<char>>& board) {
        if(board.size() == 0) return ;
        m = board.size(), n = board[0].size();
        for(int i = 0; i < m; i++) {
            dfs(i, 0, board);
            dfs(i, n - 1, board);
        }
        for(int j = 0; j < n; j++) {
            dfs(0, j, board);
            dfs(m - 1, j, board);
        }
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(board[i][j] == '*')
                    board[i][j] = 'O';
                else if(board[i][j] == 'O')
                    board[i][j] = 'X';
            }
        }
    }
    void dfs(int row, int col, vector<vector<char>>& board) {
        if(board[row][col] != 'O') return;
        board[row][col] = '*';
        if(row - 1 > 0) dfs(row - 1, col, board);
        if(col - 1 > 0) dfs(row, col - 1, board);
        if(row + 1 < m) dfs(row + 1, col, board);
        if(col + 1 < n) dfs(row, col + 1, board);
    }
};
```

134. Gas Station [Medium]

There are N gas stations along a circular route, where the amount of gas at station i is gas[i]. You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to 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：gas[i] – cost[i]为车行驶的代价。要想能走完一圈，必须时刻保持从出发点到终点始终行驶代价总和temp >= 0～假设从0开始出发，累积temp0一直是>=0的。到点A的时候发现<0了。此时可以想：从0~A的temp0小于0，而前面的0~某点的代价总和一直是>=0的，所以可以得知前面任何一点都不能作为出发点了，因为temp0 – 0~某点的代价总是小于0的不满足题意。所以只能从下一个结点开始尝试。从A+1结点开始出发，累积temp1一直是>=0的。到点B的时候发现<0了。此时可以想：从A+1 ~ B的temp1小于0，那么前面的A+1到某点的代价总和一直是>=0的，所以可以得知前面任何一点都不能作为出发点了，因为temp1 – A+1~某点的代价总是小于0不和题意的。所以就从下一点开始尝试.... 2.终于有一次可以从D开到n-1这个点了。只要看能不能车再从0开到D-1了。也就是想看temp2+temp0+temp1的过程中可不可能总代价小于0.结论是只要所有点总代价total>=0，那么一定可以开一圈。可以这么证明：total>=0 temp0 和temp1都是在最后一个点game over的 前面都是正数那么ok，最后一个数是导致<=0的点，但是去掉了temp0这个负数，总代价毫无疑问还是大于0的不用说。所以说最后只要total>=0那么就一定可以走结束一圈。啊哦- -我竟然说了这么多=_=

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        if(gas.empty())
            return -1;
        int total = 0, temp = 0, index = 0;
        for(int i = 0; i < gas.size(); i++) {
            total += gas[i] - cost[i];
            if(temp >= 0) {
                temp += gas[i] - cost[i];
            } else {
                temp = gas[i] - cost[i];
                index = i;
            }
        }
        return total >= 0 ? index : -1;
    }
};
```

136. Single Number [Easy]

Given an array of integers, every element appears twice except for one. Find that single one.

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

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        for(int i = 0; i < nums.size() - 1; i = i + 2) {
            if(nums[i] != nums[i + 1]) {
                return nums[i];
            }
        }
        return nums[nums.size() - 1];
    }
};
```

137. Single Number II [Medium]

Given an array of integers, every element appears three times except for one. Find that single one.

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

```cpp
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        for(int i = 0; i <= nums.size() - 4; i = i + 3) {
            if(nums[i] != nums[i + 2]) {
                return nums[i];
            }
        }
        return nums[nums.size() - 1];
    }
};
```

141. Linked List Cycle [Easy]

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

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

```cpp
update v2.0:
//后来想到了一个不赖皮的方法 用 set 存储 判断每次存入 next 结点后 set 的 size() 有没有发生变化
//没有发生变化说明这个结点已经被存储过了 就说明是个环
/**
 * 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 == NULL)
            return false;
        set<ListNode *> s;
        ListNode *t;
        t = head;
        int cnt = 0;
        while(t->next != NULL) {
            s.insert(t);
            if(cnt == s.size()) {
                return true;
            }
            cnt = s.size();
            t = t->next;
        }
        return false;
    }
};



version 1.0:
//感觉自己做这道题的方法有点赖皮。。。唔。。。
/**
 * 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 == NULL)
            return false;
        ListNode *p;
        p = head;
        p->val = 99999999;
        while(p->next != NULL) {
            p = p->next;
```

```
            if(p->val != 99999999) {
                p->val = 99999999;
            } else {
                return true;
            }
        }
        return false;
    }
};
```

142. Linked List Cycle II [Medium]

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.

```
/**
 * 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 == NULL) {
            return NULL;
        }
        set<ListNode *> s;
        ListNode *p = head;
        int cnt = 0;
        while(p->next != NULL) {
            s.insert(p);
            if(cnt == s.size()) {
                return p;
            }
            cnt = s.size();
            p = p->next;
        }
        return NULL;
    }
};
```

143. Reorder List [Medium]

Given a singly linked list L: L0→L1→…→Ln-1→Ln, reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.

For example, Given {1,2,3,4}, reorder it to {1,4,2,3}.

题目大意：按照L0→Ln→L1→Ln-1→L2→Ln-2→...重新排列链表～

分析：1.找到链表的中间结点mid——使用mid走一步、tail走两步的方式，直到tail走到链表尾部的时候，mid正好指向中间位置。2.从mid结点开始将后面的链表反转，返回反转后的头结点 3.p指向前面一段链表，q指向后面一段链表，将p和q合并：首先令a和b是p和q的临时结点，然后将pq分别向后移动一位，最后将a->next = b;b->next = p;

```
class Solution {
public:
    void reorderList(ListNode* head) {
        ListNode *p = head, *a = head, *b = head;
        ListNode *mid = findMidNode(head);
        ListNode *q = reverseList(mid);
        while (p != NULL && q != NULL) {
            a = p;
            b = q;
            p = p->next;
            q = q->next;
            a->next = b;
            b->next = p;
        }
        if (q != NULL) b->next = q;
    }

    ListNode* findMidNode(ListNode* head) {
        ListNode *mid = head, *tail = head, *last = head;
        while (tail != NULL && tail->next != NULL) {
            last = mid;
            mid = mid->next;
            tail = tail->next->next;
        }
        if (last != NULL) last->next = NULL;
        return mid;
    }

    ListNode* reverseList(ListNode* head) {
        if (head == NULL) return head;
        ListNode *cur = head, *pre = NULL, *temp = NULL;
        while (cur != NULL) {
            temp = cur->next;
            cur->next = pre;
            pre = cur;
            cur = temp;
        }
```

```
        return pre;
    }
};
```

144. Binary Tree Preorder Traversal [Medium]

Given a binary tree, return the preorder traversal of its nodes' values.

For example: Given binary tree {1,#,2,3}, 1 \ 2 / 3 return [1,2,3].

```
/**
 * 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) {
        stack<TreeNode*> s;
        vector<int> v;
        if(root == NULL) {
            return v;
        }
        TreeNode *p = root;
        s.push(p);
        while(!s.empty()) {
            p = s.top();
            s.pop();
            v.push_back(p->val);
            if(p->right != NULL) {
                s.push(p->right);
            }
            if(p->left != NULL) {
                s.push(p->left);
            }
        }
        return v;
    }
};
```

145. Binary Tree Postorder Traversal [Hard]

Given an input string, reverse the string word by word.

For example, Given s = “the sky is blue”, return “blue is sky the”.

分析：后序遍历，左右根～

```
/**
 * 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> result;
    vector<int> postorderTraversal(TreeNode* root) {
        dfs(root);
        return result;
    }
    void dfs(TreeNode* root) {
        if(root == NULL) return;
        dfs(root->left);
        dfs(root->right);
        result.push_back(root->val);
    }
};
```

151. Reverse Words in a String [Medium]

Given an input string, reverse the string word by word.

For example, Given s = “the sky is blue”, return “blue is sky the”.

分析：我用的方法是把字符串中的所有单词放入栈里，然后将栈里的所有字符串弹栈到字符串s中～

```
class Solution {
public:
    void reverseWords(string &s) {
        stack<string> sstack;
        int flag = 0;
        string temp = "";
        for(int i = 0; i < s.length(); i++) {
            if(s[i] != ' ' && flag == 0) {
                temp = "";
                temp += s[i];
                flag = 1;
```

```
            } else if(s[i] != ' ') {
                temp += s[i];
            } else if(s[i] == ' ' && flag == 1){
                sstack.push(temp);
                flag = 0;
            }
            if(i == s.length() - 1 && flag == 1)
                sstack.push(temp);
        }
        s = "";
        while(!sstack.empty()) {
            string temp = sstack.top();
            s += temp;
            sstack.pop();
            if(!sstack.empty())
                s += " ";
        }
    }
};
```

152. Maximum Product Subarray [Medium]

Find the contiguous subarray within an array (containing at least one number) which has the largest product. For example, given the array [2,3,-2,4], the contiguous subarray [2,3] has the largest product = 6.

题目大意：给一个整型数组，求该数组中所有连续子数组的元素乘积的最大值～

分析：需要保存临时最大值和最小值，因为最大值乘以一个正数可能构成新的最大值，而最小值乘以负数也可能构成新的最大值。result是要求的结果，maxValue为nums[i]之前的，和nums[i]相邻的乘积的最大值，minValue为nums[i]之前的，和nums[i]相邻的乘积的最小值。首先令result、maxValue和minValue都为nums[0], i从nums[1]开始一直到结束，tempMax为考虑是否选择之前的maxValue与nums[i]相乘，如果相乘结果更大就保留，否则就选择nums[i]本身为最大。tempMin同理～然后maxValue和minValue比较tempMax/tempMin与minValue * nums[i]的大小关系，maxValue取较大值，minValue取较小值～而result是取所有maxValue中的最大值～最后返回result～

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        if (nums.size() == 0) return 0;
        int result = nums[0], maxValue = nums[0], minValue = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            int tempMax = max(nums[i], maxValue * nums[i]);
            int tempMin = min(nums[i], maxValue * nums[i]);
            maxValue = max(tempMax, minValue * nums[i]);
            minValue = min(tempMin, minValue * nums[i]);
            result = max(maxValue, result);
```

```
        }
        return result;
    }
};
```

153. Find Minimum in Rotated Sorted Array [Medium]

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

You may assume no duplicate exists in the array.

分析：二分搜索法，即使旋转过了也会一半的任何一个元素始终比另一半的任何一个元素大，所以如果nums[mid] < nums[high]，说明最小元素一定在[left, mid]中，所以令high = mid；否则一定在[mid + 1, high]中，令low = mid + 1～～

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

155. Min Stack [Easy]

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.

```
class MinStack {
public:
    stack<int> s;
    stack<int> min;
```

```
    void push(int x) {
        s.push(x);
        if(min.empty() || min.top() >= x) {
            min.push(x);
        }
    }//这里 if 语句里面是 >= 的原因是，假设有两个相同的最小值8，
    //如果只存一个（只填 > ），假设两个 stack 同时pop了一个8，
    //那么其实最小值还有8但是已经被 pop 掉了

    void pop() {
        if(s.top() == min.top()) {
            s.pop();
            min.pop();
        } else {
            s.pop();
        }
    }

    int top() {
        return s.top();
    }

    int getMin() {
        return min.top();
    }
};
```

160. Intersection of Two Linked Lists [Easy]

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.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
```

```
class Solution {
public:
    int getlength(ListNode *A) {
        int len = 0;
        while(A != NULL) {
            len++;
            A = A->next;
        }
        return len;
    }

    ListNode* func(ListNode *B, int movelen) {
        while(movelen--) {
            B = B->next;
        }
        return B;
    }

    ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
        if(headA == NULL || headB == NULL) {
            return NULL;
        }
        int lenA = getlength(headA);
        int lenB = getlength(headB);
        ListNode *p, *q;
        p = headA;
        q = headB;
        if(lenA > lenB) {
            p = func(headA, lenA - lenB);
        }
        if(lenA < lenB) {
            q = func(headB, lenB - lenA);
        }
        while(p != q && p != NULL && q != NULL) {
            p = p->next;
            q = q->next;
        }
        return p;
    }
};
```

162. Find Peak Element [Medium]

A peak element is an element that is greater than its neighbors.

Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that num[-1] = num[n] = -∞.

For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.

```
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        if(nums.size() == 1 || (nums[0] > nums[1]))
            return 0;
        if(nums[nums.size() - 1] > nums[nums.size() - 2]) {
            return nums.size() - 1;
        }
        for(int i = 1; i < nums.size() - 1; i++) {
            if(nums[i] > nums[i - 1] && nums[i] > nums[i + 1]) {
                return i;
            }
        }
    }
};
```

165. Compare Version Numbers [Easy]

Compare two version numbers version1 and version2. If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character. The . character does not represent a decimal point and is used to separate number sequences. For instance, 2.5 is not “two and a half” or “half way to version three”, it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

0.1 < 1.1 < 1.2 < 13.37

```
class Solution {
public:
    int compareVersion(string version1, string version2) {
        while(!version1.empty() || !version2.empty()) {
            int ver1 = version1.empty() ? 0 : stoi(version1);
            int ver2 = version2.empty() ? 0 : stoi(version2);
            if(ver1 > ver2) {
                return 1;
            } else if(ver1 < ver2) {
                return -1;
            } else {
                int pos1 = version1.find('.');
```

```
                int pos2 = version2.find('.');
                version1 = (pos1 == string::npos) ? "" :
version1.substr(pos1 + 1);
                version2 = (pos2 == string::npos) ? "" :
version2.substr(pos2 + 1);
            }
        }
        return 0;
    }
};
```

166. Fraction to Recurring Decimal [Medium]

Given two integers representing the numerator and denominator of a fraction, return the fraction in string format.

If the fractional part is repeating, enclose the repeating part in parentheses.

For example,

Given numerator = 1, denominator = 2, return “0.5”. Given numerator = 2, denominator = 1, return “2”. Given numerator = 2, denominator = 3, return “0.(6)”. Hint:

No scary math, just apply elementary math knowledge. Still remember how to perform a long division? Try a long division on 4/9, the repeating part is obvious. Now try 4/333. Do you see a pattern? Be wary of edge cases! List out as many test cases as you can think of and test your code thoroughly.

题目大意：给定两个整型数分子和分母，以小数的形式返回它们的结果，当有循环小数时，将循环的部分用括号括起来～

分析：先忽略符号，为了防止溢出，转换为long型的a和b，以绝对值形式求a/b的结果：a/b的结果是结果的整数部分，如果余数r = a % b不等于0，说明还有小数部分～ 用map标记余数r应该在string result的哪个位置，如果map[r]不为0说明出现过，那么就在m[r]（出现的位置）的前面添加一个(，在result结尾添加一个)，表示这部分会循环～

注意点：1.如果除数等于0，直接返回0，为了避免出现返回-0的情况～2.如果(numerator < 0) ^ (denominator < 0)等于1，说明他们一正一负，结果是负数，所以要在result的第一位添加一个“-”。3.如果一开始的r不等于0，说明有小数部分，则在计算小数部分之前添加一个”.”

```
class Solution {
public:
    string fractionToDecimal(int numerator, int denominator) {
        if(numerator == 0) return "0";
        string result = "";
        if(((numerator < 0) ^ (denominator < 0)) == 1)
            result += '-';
        long a = abs((long)numerator), b = abs((long)denominator);
        result += to_string(a / b);
        long r = a % b;
```

```
        if(r != 0) result += '.';
        map<int, int> m;
        while(r != 0) {
            if(m[r] != 0) {
                result.insert(m[r], 1, '(');
                result += ')';
                break;
            }
            m[r] = result.size();
            r = r * 10;
            result += to_string(r / b);
            r = r % b;
        }
        return result;

    }
};
```

167. Two Sum II – Input array is sorted [Medium]

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 have exactly one solution.

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

分析：每个数遍历找他的后面是否存在与它相加等于target的数，如果存在就放入result数组中～

```
class Solution {
public:
    vector<int> twoSum(vector<int>& numbers, int target) {
        vector<int> result(2);
        for(int i = 0; i < numbers.size(); i++) {
            for(int j = i + 1; j < numbers.size(); j++) {
                if(numbers[i] + numbers[j] == target) {
                    result[0] = i + 1;
                    result[1] = j + 1;
                    break;
                } else if(numbers[i] + numbers[j] > target) {
                    break;
                }
            }
            if(result[0] != 0)
                break;
```

```
        }
        return result;
    }
};
```

168. Excel Sheet Column Title [Easy]

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

```
/*
把 A~Z 对应 0 ~ 25
满 26 进 1
只要看当前 n-1 的值是否 /26 != 0
如果是的 则 将 (n-1) % 26 + 'A' 存储在 s 之前
如果不是 则 退出 while 循环
*/
class Solution {
public:
    string convertToTitle(int n) {
        string s;
        while(n) {
            s = (char)((n - 1) % 26 + 'A') + s;
            n = (n - 1) / 26;
        }
        return s;
    }
};
```

169. Majority Element [Easy]

Given an array of size n, 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.

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int temp = 0, maxvalue = 0, ans = nums[0];
        for(int i = 1; i < nums.size(); i++) {
            if(nums[i - 1] == nums[i]) {
                temp++;
```

```
                if(temp > maxvalue) {
                    maxvalue = temp;
                    ans = nums[i];
                }
            } else {
                temp = 0;
            }
        }
        return ans;
    }
};
```

171. Excel Sheet Column Number [Easy]

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

```
class Solution {
public:
    int titleToNumber(string s) {
        int ans = 0;
        for(int i = 0; i < s.length(); i++) {
              ans = ans*26 + s[i] - 'A' + 1;
        }
        return ans;
    }
};
```

172. Factorial Trailing Zeroes [Easy]

Given an integer n, return the number of trailing zeroes in n!.

Note: Your solution should be in logarithmic time complexity.

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

题目大意：在logn的时间内 找到n！末尾有几个零

```
//5*2 = 10出现一个0，n*(n-1)*(n-2)...1当中能被5整除的数少于能被2整除的数
//所以能被5整除的5的个数就是0的个数
//比如25!,25 = 5 * 5有两个5，20，15，10，5各含一个5，这六个5分别和2结合相乘就能得到末尾6个0
//所以只要count每个因子中5的个数就行
```

```
5
6  class Solution {
7  public:
8      int trailingZeroes(int n) {
9          int cnt = 0;
10         while(n != 0) {
11             cnt = cnt + n / 5;
12             n = n / 5;
13         }
14         return cnt;
15     }
16 };
```

179. Largest Number [Medium]

Given a list of non negative integers, arrange them such that they form the largest number.

For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330.

Note: The result may be very large, so you need to return a string instead of an integer.

题目大意：给一个非负整数数组，求怎样把他们连接起来构成的整数最大～

分析：先把他们转换成字符串数组arr，然后对arr进行排序，排序规则是return (a + b) > (b + a);即a+b拼接起来的字符串应该比b+a拼接起来的字符串大～然后将排序后的arr数组从头到尾连接起来～

```
1  class Solution {
2  public:
3      string largestNumber(vector<int>& nums) {
4          string result = "";
5          vector<string> arr(nums.size());
6          for(int i = 0; i < nums.size(); i++)
7              arr[i] = to_string(nums[i]);
8          auto cmp = [](string a, string b) {
9              return (a + b) > (b + a);
10         };
11         sort(arr.begin(), arr.end(), cmp);
12        for(int i = 0; i < arr.size(); i++)
13             result += arr[i];
14         return result[0] == '0' ? "0" : result;
15     }
16 };
```

189. Rotate Array [Easy]

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note: Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

[show hint]

Hint: Could you do it in-place with O(1) extra space? Related problem: Reverse Words in a String II

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

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k = k % n;
        for(int i = 0; i <= (n - 1 - k) / 2; i++) {
            swap(nums[i], nums[n - 1 - k - i]);
        }
        for(int i = n - k; i <= (n - 1 + n - k) / 2; i++) {
            swap(nums[i], nums[n - 1 + n - k - i]);
        }
        for(int i = 0; i <= (n - 1) / 2; i++) {
            swap(nums[i], nums[n - 1 - i]);
        }
    }
};
```

190. Reverse Bits [Easy]

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?

```cpp
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t ans = 0;
        for(int i = 0; i < 32; i++) {
            ans = ans * 2 + n % 2;
            n = n / 2;
        }
        return ans;
    }
};
```

191. Number of 1 Bits [Easy]

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

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

```cpp
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int ans = 0;
        while(n != 0) {
            ans = ans + n % 2;
            n = n / 2;
        }
        return ans;
    }
};
```

198. House Robber [Easy]

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 @ifanchu for adding this problem and creating all test cases. Also thanks to @ts for adding additional test cases.

```cpp
class Solution {
```

```
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if(n == 0) {
            return 0;
        }
        if(n == 1) {
            return nums[0];
        }
        vector<int> dp(n, 0);
        dp[0] = nums[0];
        dp[1] = nums[0] > nums[1] ? nums[0] : nums[1];
        for(int i = 2; i < n; i++) {
            int temp = dp[i - 2] + nums[i];
            dp[i] = temp > dp[i - 1] ? temp : dp[i - 1];
        }
        return dp[n - 1];
    }
};
```

199. Binary Tree Right Side View [Medium]

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom. For example: Given the following binary tree, 1 <— / \ 2 3 <— \ \ 5 4 <— You should return [1, 3, 4].

分析：这道题可以用广度优先搜索也可以用深度优先搜索。这里用广度优先方法解决： 如果root为空，则返回空vector。建立存放TreeNode指针的队列，将root结点入队；出队root的同时入队root的存在的left和right结点；按照层序遍历的方式，把每一层的最后一个结点的值存入vector中，最后返回vector～

```
/**
 * 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> rightSideView(TreeNode* root) {
        vector<int> v;
        queue<TreeNode*> q;
        if(root == NULL)
            return v;
```

```
        q.push(root);
        TreeNode* h;
        while(!q.empty()) {
            int size = q.size();
            while(size--) {
                h = q.front();
                q.pop();
                if(h->left != NULL)
                    q.push(h->left);
                if(h->right != NULL)
                    q.push(h->right);
            }
            v.push_back(h->val);
        }
        return v;
    }
};
```

200. Number of Islands [Medium]

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

11110 11010 11000 00000 Answer: 1

Example 2:

11000 11000 00100 00011 Answer: 3

题目大意：给一个地图，0表示水，1表示陆地，计算陆地的个数

分析：计算连通分量的个数，每次遍历是陆地(grid[x][y] == 1)的区域，dfs中从4个方向遍历，每次将访问过的grid标记为'0'，main函数中进入dfs多少次就表示有多少个岛屿（也就是有多少个连通分量的个数）

```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        if (grid.size() == 0) return 0;
        n = grid.size(), m = grid[0].size();
        for (int x = 0; x < n; x++) {
            for (int y = 0; y < m; y++) {
                if (grid[x][y] == '1') {
                    dfs(x, y, grid);
                    cnt++;
                }
            }
```

```
        }
        return cnt;
    }
private:
    int n, m, cnt = 0;
    int arr[5] = {1, 0, -1, 0, 1};
    void dfs(int x, int y, vector<vector<char>>& grid) {
        grid[x][y] = '0';
        for (int i = 0; i < 4; i++) {
            int tx = x + arr[i], ty = y + arr[i+1];
            if (tx >= 0 && tx < n && ty >= 0 && ty < m && grid[tx][ty]
== '1')
                dfs(tx, ty, grid);
        }
    }
};
```

202. Happy Number [Easy]

Write an algorithm to determine if a number is “happy”.

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example: 19 is a happy number

12 + 92 = 82 82 + 22 = 68 62 + 82 = 100 12 + 02 + 02 = 1 Credits: Special thanks to @mithmatt and @ts for adding this problem and creating all test cases.

```
//即使输入的数为9*10^10，也会在第一次 while 降到3位数，第二次 while 降到2位数，不
超过10次肯定能知道是否满足题目条件
//所以加了一个 cnt 计数器 在超过10次就跳出循环说明不满足题意

class Solution {
public:
    bool isHappy(int n) {
        int cnt = 0;
        while(n != 1) {
            cnt++;
            int temp = 0;
            while(n) {
                temp = temp + (n % 10) * (n % 10);
                n = n / 10;
            }
            n = temp;
            if(cnt == 10) {
```

```
                return false;
            }
        }
        return true;
    }
};
```

203. Remove Linked List Elements [Easy]

Remove all elements from a linked list of integers that have value val.

Example Given: 1 –> 2 –> 6 –> 3 –> 4 –> 5 –> 6, val = 6 Return: 1 –> 2 –> 3 –> 4 –> 5

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

204. Count Primes [Easy]

Description:

Count the number of prime numbers less than a non-negative number, n.

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

Hint:

Let's start with a isPrime function. To determine if a number is prime, we need to check if it is not divisible by any number less than n. The runtime complexity of isPrime function would be O(n) and hence counting the total prime numbers up to n would be O(n2). Could we do better?

As we know the number must not be divisible by any number > n / 2, we can immediately cut the total iterations half by dividing only up to n / 2. Could we still do better?

Let's write down all of 12's factors:

2 × 6 = 12 3 × 4 = 12 4 × 3 = 12 6 × 2 = 12 As you can see, calculations of 4 × 3 and 6 × 2 are not necessary. Therefore, we only need to consider factors up to √n because, if n is divisible by some number p, then n = p × q and since p ≤ q, we could derive that p ≤ √n.

Our total runtime has now improved to O(n1.5), which is slightly better. Is there a faster approach?

```
class Solution {
public:
    int countPrimes(int n) {
        vector<int> book(n, 1);
        for(int i = 2; i * i < n; i++) {
            if(book[i] == 1)
                for(int j = i * i; j < n; j = j + i)
                    book[j] = 0;
        }
        int cnt = 0;
        for(int i = 2; i < n; i++) {
            if(book[i] == 1)
                cnt++;
        }
        return cnt;
    }
};
```

205. Isomorphic Strings [Easy]

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

For example, Given "egg", "add", return true.

Given “foo”, “bar”, return false.

Given “paper”, “title”, return true.

Note: You may assume both s and t have the same length.

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        int len = s.length();
        int lent = t.length();
        if(len != lent)
            return false;
        map<char, char> stot;
        map<char, char> ttos;
        for(int i = 0; i < len; i++) {
            if(stot.find(s[i]) != stot.end() && stot[s[i]] != t[i]
            || ttos.find(t[i]) != ttos.end() && ttos[t[i]] != s[i])
                return false;
            stot[s[i]] = t[i];
            ttos[t[i]] = s[i];
        }
        return true;
    }
};
```

206. Reverse Linked List [Easy]

Reverse a singly linked list.

题目大意：反转一个单链表～

分析：设立三个指针：cur——当前结点；pre——当前结点的前一个结点；temp——临时结点（标记cur的next）。首先保存cur的next到temp，然后将cur的next指向pre，将pre移动到当前cur，然后将cur指向temp，直到cur==NULL，返回pre即反转了该链表～

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if (head == NULL) return head;
        ListNode *cur = head, *pre = NULL, *temp = NULL;
        while (cur != NULL) {
            temp = cur->next;
            cur->next = pre;
            pre = cur;
            cur = temp;
        }
```

```
        return pre;
    }
};
```

209. Minimum Size Subarray Sum [Medium]

Given an array of n positive integers and a positive integer s, find the minimal length of a contiguous subarray of which the sum ≥ s. If there is not one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and s = 7, the subarray [4,3] has the minimal length under the problem constraint.

More practice: If you have figured out the O(n) solution, try coding another solution of which the time complexity is O(n log n).

分析：sum为前i个数的和，长度比nums多一个，sum[0] = 0。这样从0开始一直到len，遍历sum计算sum[j]与sum[i]的差大于等于s的时候的j-i长度，把它与minlen比较，如果比minlen小就更新minlen。 一开始minlen的初始化值是len + 1，如果最后minlen的值仍旧为len + 1，说明没有找到这样的minlen满足题意。则直接返回0；否则返回minlen的值～～

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int len = nums.size();
        int minlen = len + 1;
        vector<int> sum(len + 1);
        for(int i = 1; i <= len; i++) {
            sum[i] = sum[i-1] + nums[i-1];
        }
        for(int i = 0; i <= len; i++) {
            for(int j = i + 1; j <= len; j++) {
                if(sum[j] - sum[i] >= s) {
                    minlen = min(minlen, j - i);
                    break;
                }
            }
        }
        if(minlen == len + 1)
            return 0;
        return minlen;
    }
};
```

213. House Robber II [Medium]

Note: This is an extension of House Robber. After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street. 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.

分析：这题是上一题House Robber的升级版～～新加了环形的街道biu biu biu～～ 所以就会考虑到，最后一个和第一个房子是不能够同时进入的~要不然会告诉警察叔叔～～ 所以分为两种情况～～ 0.不包括最后一个屋子～就抢劫0~n-2号屋子～ 1.不包括第一个屋子～就抢劫1~n-1号屋子～ 这样的话，return上面两种情况的最大值就好了～调用两次子函数求值，主函数取其最大值返回～

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

215. Kth Largest Element in an Array [Medium]

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example, Given [3,2,1,5,6,4] and k = 2, return 5.

Note: You may assume k is always valid, 1 ≤ k ≤ array's length.

分析：一开始用set，后来发现是“not the kth distinct element”，所以说不能用set，set里面元素必须是不相同的，所以这里用multiset就能解决。 因为multiset会自动排序，所以每次将数字放入multiset，如果集合里面容量超过k个就把最小的那个移除～ 到最后输出*s.begin()即为第k大～

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        multiset<int> s;
        for(int i = 0; i < nums.size(); i++) {
            s.insert(nums[i]);
            if(s.size() > k)
                s.erase(s.begin());
        }
        return *s.begin();
    }
};
```

216. Combination Sum III [Medium]

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.
Example 1:

Input: k = 3, n = 7

Output:

[[1,2,4]]

Example 2:

Input: k = 3, n = 9

Output:

[[1,2,6], [1,3,5], [2,3,4]]

分析：一个dfs回溯解决，如果当前k == 0而且 n == 0说明一定达到了k个数，和为n，则将当前path压入result数组中；从start开始一直到9，分别压入path中深度优先搜索。深度优先搜索完了之后记得回溯path中pop出最后一个元素～ 因为题目要求结果集合必须是按照集合顺序，也就是从小到大而且没有重复元素，那么就要设立一个start变量，每次for循环的时候从start开始，一开始start为0，每次规定start为i+1，即只能从当前数字的下一个数字开始，这样就能保证结果是递增无重复数字的集合序列～～

```
class Solution {
public:
    vector<vector<int>> result;
    vector<int> path;
```

```
    vector<vector<int>> combinationSum3(int k, int n) {
        dfs(k, n, 1);
        return result;
    }

    void dfs(int k, int n, int start) {
        if(k == 0) {
            if(n == 0)
                result.push_back(path);
            return ;
        }
        for(int i = start; i <= 9; i++) {
            path.push_back(i);
            dfs(k - 1, n - i, i + 1);
            path.pop_back();
        }
    }
};
```

217. Contains Duplicate [Easy]

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        set<int> m;
        for(int i = 0; i < nums.size(); i++) {
            m.insert(nums[i]);
        }
        return nums.size() != m.size();
    }
};
```

219. Contains Duplicate II [Easy]

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the difference between i and j is at most k.

```
class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        set<int> s;
```

```
        int t = 0;
        for(int i = 0; i < nums.size(); i++) {
            s.insert(nums[i]);
            if(s.size() == t) {
                for(int j = i - 1; j >= 0 && j >= i - k; j--) {
                    if(nums[i] == nums[j])
                        return true;
                }
            }
            t = s.size();
        }
        return false;
    }
};
```

220. Contains Duplicate III [Medium]

Given an array of integers, find out whether there are two distinct indices i and j in the array such that the absolute difference between nums[i] and nums[j] is at most t and the absolute difference between i and j is at most k.

题目大意：给一个整数数组，找到是否存在两个不同的下标i和j，使得nums[i]和nums[j]的差的绝对值不超过t并且i和j的差的绝对值不超过k～

分析：建立一个map，对应的是元素的值到元素的下标的映射。指针i将nums数组从头遍历到尾，j指针一开始指向0。i向右走的时候如果i和j之差大于k，且m中有nums[j]，就将nums[j]从m中移除，且j向前走一步～这样就保证了m中所有的元素满足第一个条件：i和j的差的绝对值不超过k～

接下来考虑nums[i]和nums[j]的差的绝对值不超过t，abs(num[i] – nums[j]) <= t 则 nums[j]的最小可能满足条件的值为>=nums[i] – t的，所以使用map中的lower_bound，寻找第一个大于等于nums[i] – t的地方，找到后标记为a，此时的a只是取到了可能满足的最小的a，但(a – nums[i])不一定满足，所以检验a是否存在于map中且是否abs(a->first – nums[i]) <= t。如果都满足说明可以return true～

为什么要循环的最后写map的插入呢，因为比较的是i之前的所有元素～为了防止找到的是nums[i]本身，然后让nums[i]自己本身和自己比较差值了，这样结果就不对了～ 如果到最后都没有能够return true，则return false～

```
class Solution {
public:
    bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t)
{
        map<long, int> m;
        int j = 0;
        for (int i = 0; i < nums.size(); ++i) {
            if (i - j > k && m[nums[j]] == j) m.erase(nums[j++]);
            auto a = m.lower_bound((long)nums[i] - t);
            if (a != m.end() && abs(a->first - nums[i]) <= t) return
true;
```

```
            m[nums[i]] = i;
        }
        return false;
    }
};
```

222. Count Complete Tree Nodes [Medium]

Given a complete binary tree, count the number of nodes.

Definition of a complete binary tree from Wikipedia: In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

题目大意：给一个完全二叉树，计算结点的个数～

分析：完全二叉树满足当最左结点和最右结点等高(h)的时候，结点数为2的h次方-1；否则等于左子树的结点数 + 右子树的结点数 + 1。且完全二叉树的左子树和右子树也是一个完全二叉树～

```
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (root == NULL) return 0;
        TreeNode *l = root, *r = root;
        int cntLeft = 0, cntRight = 0;
        while (l != NULL) {
            l = l->left;
            cntLeft++;
        }
        while (r != NULL) {
            r = r->right;
            cntRight++;
        }
        if (cntLeft == cntRight) return pow(2, cntLeft) - 1;
        return countNodes(root->left) + countNodes(root->right) + 1;
    }
};
```

223. Rectangle Area [Easy]

Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.

Rectangle Area Assume that the total area is never beyond the maximum possible value of int.

```
class Solution {
```

```
public:
    int computeArea(int A, int B, int C, int D, int E, int F, int G,
int H) {
        int maxae = A > E ? A : E;
        int mincg = C > G ? G : C;
        int maxfb = F > B ? F : B;
        int minhd = H > D ? D : H;
        int overlap;
        if(maxae >= mincg || maxfb >= minhd)
            overlap = 0;
        else
            overlap = (mincg - maxae) * (minhd - maxfb);
        return (C - A) * (D - B) + (G - E) * (H - F) - overlap;
    }
};
```

225. Implement Stack using Queues [Easy]

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). Update (2015-06-11): The class name of the Java function had been updated to MyStack instead of Stack.

```
class Stack {
public:
    // Push element x onto stack.
    queue<int> q1, q2;
    void push(int x) {
        q1.push(x);
    }

    // Removes the element on top of the stack.
    void pop() {
        while(q1.size() != 1) {
            int t = q1.front();
            q1.pop();
            q2.push(t);
        }
        q1.pop();
        while(!q2.empty()) {
            int t = q2.front();
```

```
            q2.pop();
            q1.push(t);
        }
    }

    // Get the top element.
    int top() {
        return q1.back();
    }

    // Return whether the stack is empty.
    bool empty() {
        return q1.empty();
    }
};
```

226. Invert Binary Tree [Easy]

Invert a binary tree.

4 / \ 2 7 / \ / \ 1 3 6 9 to 4 / \ 7 2 / \ / \ 9 6 3 1 Trivia: This problem was inspired by this original tweet by Max Howell: Google: 90% of our engineers use the software you wrote (Homebrew), but you can’t invert a binary tree on a whiteboard so fuck off.

```
/**
 * 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* invertTree(TreeNode* root) {
        TreeNode* p = root;
        if(root == NULL) {
            return NULL;
        }

        TreeNode* temp = root->left;
        root->left = root->right;
        root->right = temp;

        invertTree(root->left);
        invertTree(root->right);

```

```
        return p;
    }
};
```

228. Summary Ranges [Medium]

Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

分析：判断当前nums[i]与nums[i-1]是否是相连，如果是相连就令flag = 1，如果不相连了就将前面的结果放入result数组中。最后for循环之外还要记得把temp字符串再压入数组result中，因为当前最后一次的temp还未被处理。最后返回结果～

```
class Solution {
public:
    vector<string> summaryRanges(vector<int>& nums) {
        vector<string> result;
        if(nums.size() == 0)
            return result;
        string temp = "";
        int flag = 0;
        temp += to_string(nums[0]);
        for(int i = 1; i < nums.size(); i++) {
            if(nums[i] != nums[i-1] + 1 && flag == 1) {
                flag = 0;
                temp += "->" + to_string(nums[i-1]);
                result.push_back(temp);
                temp = "" + to_string(nums[i]);
            } else if(nums[i] != nums[i-1] + 1) {
                result.push_back(temp);
                temp = "" + to_string(nums[i]);
            } else {
                flag = 1;
            }
        }
        if(flag == 1) {
            temp += "->" + to_string(nums[nums.size() - 1]);
        }
        result.push_back(temp);
        return result;
    }
};
```

230. Kth Smallest Element in a BST [Medium]

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.

Note: You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Follow up: What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

Hint:

Try to utilize the property of a BST. What if you could modify the BST node's structure? The optimal runtime complexity is O(height of BST).

题目大意：给一个二叉搜索树，返回它的第k小的数～

分析：二叉搜索树的中序遍历是从小到大排序的，将前k个遍历结果放入nums中，当nums.size() >= k的时候返回nums[k-1]～

```
class Solution {
public:
    vector<int> nums;
    int kthSmallest(TreeNode* root, int k) {
        if (root == NULL || nums.size() >= k) return nums[k - 1];
        if (root->left != NULL) kthSmallest(root->left, k);
        nums.push_back(root->val);
        if (root->right != NULL) kthSmallest(root->right, k);
        return nums[k - 1];
    }
};
```

231. Power of Two [Easy]

Given an integer, write a function to determine if it is a power of two.

```
update v2.0:
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if(n <= 0)
            return false;
        return pow(2, (round)(log(n) / log(2))) == n;
    }
};


//如果没有考虑n <= 0就会超时。。
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if(n <= 0)
```

```
            return false;
        while(n != 1) {
            if(n % 2 == 0) {
                n = n / 2;
            } else {
                return false;
            }
        }
        return true;
    }
};
```

232. Implement Queue using Stacks [Easy]

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).

```
//用两个 stack 实现队列的操作
//压入s1的元素是倒序的，把它们逐个弹出再压入s2就是正序的了
class Queue {
public:
    stack<int> s1, s2;
    // Push element x to the back of queue.
    void push(int x) {
        s1.push(x);
    }

    // Removes the element from in front of queue.
    void pop(void) {
        if(s2.empty()) {
            while(!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        s2.pop();
    }

    // Get the front element.
    int peek(void) {
```

```
        if(s2.empty()) {
            while(!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        return s2.top();
    }

    // Return whether the queue is empty.
    bool empty(void) {
        if(s1.empty() && s2.empty()) {
            return true;
        } else {
            return false;
        }
    }
};
```

234. Palindrome Linked List [Easy]

Given a singly linked list, determine if it is a palindrome.

Follow up: Could you do it in O(n) time and O(1) space?

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

    ListNode* reverseList(ListNode* head) {
        if(head == NULL || head->next == NULL)
            return head;
        ListNode *newhead = head;
```

```
        while(newhead->next != NULL) {
            newhead = newhead->next;
        }
        func(head);
        return newhead;
    }

    bool isPalindrome(ListNode* head) {
        if(head == NULL || head->next == NULL)
            return true;
        ListNode *p, *q;
        p = head;
        q = head;
        while(q != NULL) {
            p = p->next;
            q = q->next;
            if(q != NULL) {
                q = q->next;
            }
        }
        p = reverseList(p);
        while(p != NULL) {
            if(head->val != p->val)
                return false;
            else {
                head = head->next;
                p = p->next;
            }
        }
        return true;
    }
};
```

235. Lowest Common Ancestor of a Binary Search Tree [Easy]

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself).”

___6__ / \ _**2** _**8** / \ / \ 0 _4 7 9 / \ 3 5 For example, the lowest common ancestor (LCA) of nodes 2 and 8 is 6. Another example is LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

```
/**
 * 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* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root==NULL || p==NULL || q==NULL)
            return NULL;
        if((p->val >= root->val && q->val <= root ->val) || (p->val <= root->val && q->val >= root->val)) {
            return root;
        }
        if(p->val > root->val && q->val > root->val) {
            return lowestCommonAncestor(root->right, p, q);
        }
        if(p->val < root->val && q->val < root->val) {
            return lowestCommonAncestor(root->left, p, q);
        }
    }
};
```

237. Delete Node in a Linked List [Easy]

237. Delete Node in a Linked List Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

分析：因为删除结点本来的步骤是找到当前结点的前一个结点，然后修改前一个结点的 next 指针指向。现在只知道当前要删除的结点，而不知道要删除结点的前一个结点，所以可以通过修改当前要删除结点的值，然后将当前结点指针指向 next 的 next （也就是变成了把后一个结点的值赋给当前结点，然后删除要删除结点的下一个结点），达到删除结点的目的～

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

238. Product of Array Except Self [Medium]

Given an array of n integers where n > 1, nums, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

Solve it without division and in O(n).

For example, given [1,2,3,4], return [24,12,8,6].

Follow up: Could you solve it with constant space complexity? (Note: The output array does not count as extra space for the purpose of space complexity analysis.)

```
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        vector<int> v(nums.size());
        int right = 1;
        v[0] = 1;
        //左边所有数字的乘积
        for(int i = 1; i < nums.size(); i++) {
           v[i] = nums[i - 1] * v[i - 1];
        }
        for(int i = nums.size() - 2; i >= 0; i--) {
            right = right * nums[i + 1];
            v[i] = v[i] * right;
        }
        return v;
    }
};
```

240. Search a 2D Matrix II [Medium]

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right. Integers in each column are sorted in ascending from top to bottom. For example,

Consider the following matrix:

[[1, 4, 7, 11, 15], [2, 5, 8, 12, 19], [3, 6, 9, 16, 22], [10, 13, 14, 17, 24], [18, 21, 23, 26, 30]] Given target = 5, return true.

Given target = 20, return false.

题目大意：编写一个有效的算法，在m×n矩阵中搜索一个值。 该矩阵具有以下属性： 每行中的整数从左到右依次排列。每列中的整数按从上到下的顺序排列。

分析：对每一行进行binary_serch()，如果能够找到则return true，否则返回false

```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        for(int i = 0; i < matrix.size(); i++)
            if (binary_search(matrix[i].begin(), matrix[i].end(),
target)) return true;
        return false;
    }
};
```

242. Valid Anagram [Easy]

Given two strings s and t, write a function to determine if t is an anagram of s.

For example, s = “anagram”, t = “nagaram”, return true. s = “rat”, t = “car”, return false.

Note: You may assume the string contains only lowercase alphabets.

Follow up: What if the inputs contain unicode characters? How would you adapt your solution to such case?

```cpp
class Solution {
public:
    bool isAnagram(string s, string t) {
        int a[26] = {0};
        if(s.length() != t.length())
            return 0;
        for(int i = 0; i < s.length(); i++) {
            a[s[i] - 'a']++;
        }
        for(int i = 0; i < t.length(); i++) {
            a[t[i] -'a']--;
        }
        for(int i = 0; i < 26; i++) {
            if(a[i] != 0) {
                return 0;
            }
        }
        return 1;
    }
};
```

258. Add Digits [Easy]

Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.

For example:

Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.

Follow up: Could you do it without any loop/recursion in O(1) runtime?

```
class Solution {
public:
    int addDigits(int num) {
        return num - (num - 1) / 9 * 9;
    }
};
```

260. Single Number III [Medium]

Given an array of numbers nums, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

For example:

Given nums = [1, 2, 1, 3, 2, 5], return [3, 5].

Note: The order of the result is not important. So in the above example, [5, 3] is also correct. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

```
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<int> v(2);
        int flag = 0;
        for(int i = 0; i < nums.size() - 1; i = i + 2) {
            if(nums[i] != nums[i + 1]) {
                v[0] = nums[i];
                for(int j = i + 1; j < nums.size() - 1; j = j + 2) {
                    if(nums[j] != nums[j + 1]) {
                        v[1] = nums[j];
                        flag = 1;
                        break;
                    }
                }
                break;
            }
        }
        if(flag == 0) {
            v[1] = nums[nums.size() - 1];
        }
        return v;
```

```
    }
};
```

263. Ugly Number [Easy]

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 6, 8 are ugly while 14 is not ugly since it includes another prime factor 7.

Note that 1 is typically treated as an ugly number.

```
class Solution {
public:
    bool isUgly(int num) {
        if(num <= 0) {
            return false;
        }
        int flag = 0;
        while(num != 1) {
            flag = 0;
            if(num % 2 == 0) {
                num = num / 2;
                flag = 1;
            }
            if(num % 3 == 0) {
                 num = num / 3;
                 flag = 1;
            }
            if(num % 5 == 0) {
                num = num / 5;
                flag = 1;
            }
            if(flag == 0) {
                return false;
            }
        }
        return true;
    }
};
```

268. Missing Number [Medium]

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

For example, Given nums = [0, 1, 3] return 2.

Note: Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int *book = new int [nums.size() + 1];
        for(int i = 0; i <= nums.size(); i++) {
            book[i] = 0;
        }
        for(int i = 0; i < nums.size(); i++) {
            book[nums[i]] = 1;
        }
        for(int i = 0; i <= nums.size(); i++) {
            if(book[i] == 0) {
                return i;
            }
        }
    }
};
```

278. First Bad Version [Easy]

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

```
// Forward declaration of isBadVersion API.
bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
        int low = 1, high = n;
        while(low <= high) {
            int mid = (high - low) / 2 + low;
            if(isBadVersion(mid)) {
                high = mid - 1;
            } else {
```

```
                low = mid + 1;
            }
        }
        return low;
    }
};
```

279. Perfect Squares [Medium]

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n. For example, given n = 12, return 3 because 12 = 4 + 4 + 4; given n = 13, return 2 because 13 = 4 + 9.

分析：建立一个n+1长度的数组dp，dp[i]表示i这个数构成平方和需要数字的最小个数。 当j*j<i的时候，temp中保存j从1开始每加1得到的dp[i-j*j] + 1的最小值～ 当j*j=i的时候，dp[i]的值就直接为1～ 从2一直到n可以计算出dp[i]的所有值～ 最后return dp[n]的值～

```
class Solution {
public:
    int numSquares(int n) {
        vector<int> dp(n+1);
        dp[1] = 1;
        for(int i = 2; i <= n; i++) {
            int temp = 99999999;
            for(int j = 1; j * j <= i; j++) {
                if(j * j == i) {
                    temp = 1;
                    break;
                }
                temp = min(temp, dp[i-j*j] + 1);
            }
            dp[i] = temp;
        }
        return dp[n];
    }
};
```

283. Move Zeroes [Easy]

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

For example, given nums = [0, 1, 0, 3, 12], after calling your function, nums should be [1, 3, 12, 0, 0].

Note: You must do this in-place without making a copy of the array. Minimize the total number of operations.

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

290. Word Pattern [Easy]

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Examples: pattern = “abba”, str = “dog cat cat dog” should return true. pattern = “abba”, str = “dog cat cat fish” should return false. pattern = “aaaa”, str = “dog cat cat dog” should return false. pattern = “abba”, str = “dog dog dog dog” should return false. Notes: You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

```cpp
class Solution {
public:
    bool wordPattern(string pattern, string str) {
        map<char, string> ptos;
        map<string, char> stop;
        int len = pattern.length();
        string *s = new string [len];
        int cnt = 0;
        for(int i = 0; i < str.length(); i++) {
            if(str[i] == ' ')
                cnt++;
        }
        if(cnt != len - 1) {
            return false;
        }
        int t = 0;
        for(int i = 0; i < str.length(); i++) {
            if(str[i] != ' ') {
```

```
                s[t] += str[i];
            } else {
                t++;
            }
        }
        for(int i = 0; i < len; i++) {
            if(ptos.find(pattern[i]) != ptos.end() && ptos[pattern[i]]
!= s[i]
            || stop.find(s[i]) != stop.end() && stop[s[i]] !=
pattern[i])
                return false;
            ptos[pattern[i]] = s[i];
            stop[s[i]] = pattern[i];
        }
        return true;
    }
};
```

292. Nim Game [Easy]

You are playing the following Nim Game with your friend: There is a heap of stones on the table, each time one of you take turns to remove 1 to 3 stones. The one who removes the last stone will be the winner. You will take the first turn to remove the stones.

Both of you are very clever and have optimal strategies for the game. Write a function to determine whether you can win the game given the number of stones in the heap.

For example, if there are 4 stones in the heap, then you will never win the game: no matter 1, 2, or 3 stones you remove, the last stone will always be removed by your friend.

思路： 设甲乙一人一次为一轮。进行了很多轮之后，让甲选的时候，如果是1，2，3那就可以通过。如果是4一定不能赢，所以如果是5，6，7可以想办法分别取1，2，3让乙来一定不能赢，所以5，6，7甲是可以赢的。 如果是8，则无论甲选任何个数，都能让乙来面临7，6，5这些必赢的选项所以甲一定输。 如果是9，10，11，则甲可以通过让乙来面临8来一定输。 如果是12，则甲无论选取任何个数，都能让乙面临9，10，11这样的一定可以赢的数字，所以12让甲必定输。 以此类推发现规律，在4或者4的倍数的时候，甲无论怎样一定输。 所以就简单一句：return n % 4

```
class Solution {
public:
    bool canWinNim(int n) {
        return n % 4;
    }
};
```

299. Bulls and Cows [Easy]

You are playing the following Bulls and Cows game with your friend: You write down a number and ask your friend to guess what the number is. Each time your friend makes a guess, you provide a hint that indicates how many digits in said guess match your secret number exactly in both digit and position (called “bulls”) and how many digits match the secret number but locate in the wrong position (called “cows”). Your friend will use successive guesses and hints to eventually derive the secret number.

For example:

Secret number: “1807” Friend’s guess: “7810” Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.) Write a function to return a hint according to the secret number and friend’s guess, use A to indicate the bulls and B to indicate the cows. In the above example, your function should return “1A3B”.

Please note that both secret number and friend’s guess may contain duplicate digits, for example:

Secret number: “1123” Friend’s guess: “0111” In this case, the 1st 1 in friend’s guess is a bull, the 2nd or 3rd 1 is a cow, and your function should return “1A1B”. You may assume that the secret number and your friend’s guess only contain digits, and their lengths are always equal.

```
class Solution {
public:
    string getHint(string secret, string guess) {
        int bull = 0, cow = 0;
        int s[10], g[10];
        memset(s, 0, sizeof(int) * 10);
        memset(g, 0, sizeof(int) * 10);
        for(int i = 0; i < secret.length(); i++) {
            if(secret[i] == guess[i]) {
                bull++;
            } else {
                s[secret[i] - '0']++;
                g[guess[i] - '0']++;
            }
        }

        for(int i = 0; i < 10; i++) {
            if(s[i] >= g[i]) {
                cow = cow + g[i];
            } else {
                cow = cow + s[i];
            }
        }

        string b = "";
        if(bull == 0)
            b = "0";
```

```
        while(bull) {
            char temp1 = (bull % 10 + '0');
            bull = bull / 10;
            b = temp1 + b;
        }

        string c = "";
        if(cow == 0)
            c = "0";
        while(cow) {
            char temp2 = (cow % 10 + '0');
            cow = cow / 10;
            c = temp2 + c;
        }

        string ans = b + 'A' + c + 'B';
        return ans;
    }
};
```

300. Longest Increasing Subsequence [Medium]

Given an unsorted array of integers, find the length of longest increasing subsequence. For example, Given [10, 9, 2, 5, 3, 7, 101, 18], The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length. Your algorithm should run in O(n2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

update v2.0: //更高效的O(nlogn)方法～ 维护一个数组vector<int> v～ 将nums[0]加入数组，对于每一个来的数nums[i]，如果大于v.back()，就将它push入数组～ 否则，就找到第一个比这个数大的位置，将该位置的数字替换为nums[i]～～ 最终返回v数组的长度~~~就是最长字串的长度啦～

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = nums.size();
        if(n == 0)
            return 0;
        vector<int> v;
        v.push_back(nums[0]);
        for(int i = 1; i < n; i++) {
            if(nums[i] > v.back()) {
                v.push_back(nums[i]);
            } else {
                *lower_bound(v.begin(), v.end(), nums[i]) = nums[i];
            }
        }
```

```
        return v.size();
    }
};
```

303. Range Sum Query – Immutable [Easy]

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

Example: Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1 sumRange(2, 5) -> -1 sumRange(0, 5) -> -3 Note: You may assume that the array does not change. There are many calls to sumRange function.

```
class NumArray {
private:
    vector<int> v;
public:
    NumArray(vector<int> &nums) {
        if(nums.size() == 0)
            return ;
        v.push_back(nums[0]);
        for(int i = 1; i < nums.size(); i++) {
            v.push_back(v[i - 1] + nums[i]);
        }
    }

    int sumRange(int i, int j) {
        if(i == 0)
            return v[j];
        return v[j] - v[i - 1];
    }
};


// Your NumArray object will be instantiated and called as such:
// NumArray numArray(nums);
// numArray.sumRange(0, 1);
// numArray.sumRange(1, 2);
```

304. Range Sum Query 2D – Immutable [Medium]

Given a 2D matrix matrix, find the sum of the elements inside the rectangle defined by its upper left corner (row1, col1) and lower right corner (row2, col2). Example: Given matrix = [[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5], [4, 1, 0, 1, 7], [1, 0, 3, 0, 5]] sumRegion(2, 1, 4, 3) -> 8 sumRegion(1, 1, 2, 2) -> 11 sumRegion(1, 2, 2, 4) -> 12 Note: You may assume that the matrix does not change. There are many calls to sumRegion function. You may assume that row1 ≤ row2 and col1 ≤ col2.

啊喂，不要看下面代码这么长就跑啊。。。。逻辑还是十分简明清晰的。。。先用一个和方阵一样大小的方阵存储入所有它的左上角所有方块上数字的总和～～然后每次调用只要大方块减去两个多余的小方块再加上重复减去的小方块就是要求的总和啦～～还是很好想象滴～～记得要注意边界条件matrix.empty()、i == 0、j == 0的时候～～当时写了很多代码写完就直接submit solution竟然直接AC了。。。//#这也能AC系列#

```
class NumMatrix {
public:
    vector<vector<int>> v;
    NumMatrix(vector<vector<int>> &matrix) {
        int m = matrix.size();
        if(matrix.empty())
            return ;
        int n = matrix[0].size();
        v = vector<vector<int>> (m, vector<int>(n));
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(i == 0 && j == 0) {
                    v[i][j] = matrix[0][0];
                } else if(i == 0) {
                    v[i][j] = v[i][j-1] + matrix[i][j];
                } else if(j == 0) {
                    v[i][j] = v[i-1][j] + matrix[i][j];
                } else {
                    v[i][j] = v[i-1][j] + v[i][j-1] + matrix[i][j] - v[i-1][j-1];
                }
            }
        }
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        if(row1 == 0 && col1 == 0) {
            return v[row2][col2];
        } else if(row1 == 0) {
            return v[row2][col2] - v[row2][col1-1];
        } else if(col1 == 0) {
            return v[row2][col2] - v[row1-1][col2];
        } else {
            return v[row2][col2] - v[row1-1][col2] - v[row2][col1-1] + v[row1-1][col1-1];
        }
    }
};


// Your NumMatrix object will be instantiated and called as such:
```

```
// NumMatrix numMatrix(matrix);
// numMatrix.sumRegion(0, 1, 2, 3);
// numMatrix.sumRegion(1, 2, 3, 4);
```

307. Range Sum Query – Mutable [Medium]

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

The update(i, val) function modifies nums by updating the element at index i to val. Example: Given nums = [1, 3, 5]

sumRange(0, 2) -> 9 update(1, 2) sumRange(0, 2) -> 8 Note: The array is only modifiable by the update function. You may assume the number of calls to update and sumRange function is distributed evenly.

题目大意：给一个整型数组，当调用sumRange(i, j)的时候返回i～j之间的元素的总和，当调用update(i, val)的时候将nums[i]的值更新为val

分析：本来想用sum[]数组标记和，然后更新sum[]直接返回的，结果还是意料之中的超时了。。用树状数组的方法可以解决～

解释：树状数组本质上是按照二分对数组进行分组，维护和查询都是O(lgn)的复杂度 树状数组与线段树：树状数组和线段树很像，但能用树状数组解决的问题，基本上都能用线段树解决，而线段树能解决的树状数组不一定能解决。相比较而言，树状数组效率要高很多～关于lowbit：lowbit = x & (-x), lowbit(x)也可以理解为能整除x的最大的2的幂次，如果要求[x, y]之内的数的和，可以转换成getsum(y) – getsum(x – 1)来解决～

```
class NumArray {
public:
    NumArray(vector<int> nums) {
       size = nums.size();
       num = vector<int>(size + 1, 0);
       sum = vector<int>(size + 1, 0);
       for(int i = 0;i < size; i++)
           update(i, nums[i]);
    }

    void update(int i, int val) {
        int old = num[i+1];
        for(int j = i + 1;j <= size; j += (j & (-j)))
            sum[j] = sum[j] - old + val;
        num[i+1] = val;
    }

    int sumRange(int i, int j) {
        return getSum(j + 1) - getSum(i);
    }
private:
```

```
    int getSum(int k) {
        int result = 0;
        for(int i = k;i > 0;i -= (i & (-i)))
            result += sum[i];
        return result;
    }
    int size;
    vector<int> sum, num;
};

/**
 * Your NumArray object will be instantiated and called as such:
 * NumArray obj = new NumArray(nums);
 * obj.update(i,val);
 * int param_2 = obj.sumRange(i,j);
 */
```

318. Maximum Product of Word Lengths [Medium]

Given a string array words, find the maximum value of length(word[i]) * length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1: Given [“abcw”, “baz”, “foo”, “bar”, “xtfn”, “abcdef”] Return 16 The two words can be “abcw”, “xtfn”.

Example 2: Given [“a”, “ab”, “abc”, “d”, “cd”, “bcd”, “abcd”] Return 4 The two words can be “ab”, “cd”.

Example 3: Given [“a”, “aa”, “aaa”, “aaaa”] Return 0 No such pair of words.

题目大意：给一个string数组，每一个string都是一个单词，只包含小写字母，求问没有相同字母的两个单词的长度乘积最大值是多少～如果不存在这样的两个单词，返回0～

分析：其实只需要知道每个单词包含哪些字母就可以了～反正一共只有26个字母，而整型长度是32位，这样就可以用一个int的32位来体现一个字母有哪些字母～比如说一个单词包含字母b，那么b – a = 1，就把从右往左的第1位置为1～等到把一个单词的所有字母相应位都置为1后，这个整型的值与另一个单词的整型值做“&”运算就可以得知他俩有没有相同的字符了～因为如果做“&”运算后的结果是0，表示没有二进制中的一位同时是1，表示这两个单词是没有相同字符的～这样就能很快判断出最大的长度乘积了～

```
class Solution {
public:
    int maxProduct(vector<string>& words) {
        vector<int> v(words.size());
        int result = 0;
        for(int i = 0; i < words.size(); i++)
            for(int j = 0; j < words[i].length(); j++)
                v[i] = v[i] | 1 << (words[i][j] - 'a');
```

```
        for(int i = 0; i < words.size(); i++)
            for(int j = i + 1; j < words.size(); j++)
                if((v[i] & v[j]) == 0)
                    result = max(result, (int)(words[i].length() *
words[j].length()));
        return result;
    }
};
```

319. Bulb Switcher [Medium]

319. Bulb Switcher My Submissions QuestionEditorial Solution Total Accepted: 17314 Total Submissions: 42815 Difficulty: Medium There are n bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the ith round, you toggle every i bulb. For the nth round, you only toggle the last bulb. Find how many bulbs are on after n rounds.

Example:

Given n = 3.

At first, the three bulbs are [off, off, off]. After first round, the three bulbs are [on, on, on]. After second round, the three bulbs are [on, off, on]. After third round, the three bulbs are [on, off, off].

So you should return 1, because there is only one bulb is on.

分析：可知题意是想判断1~n中那些数的因子个数是奇数，一个数 a = b x c 当 b = c 的时候 a 的因子个数就是奇数，所以就是判断1~n当中有多少个是某数的平方～

```
class Solution {
public:
    int bulbSwitch(int n) {
        int ans = 0;
        for(int i = 1; i * i <= n; i++) {
            ans++;
        }
        return ans;
    }
};
```

322. Coin Change [Medium]

You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Example 1: coins = [1, 2, 5], amount = 11 return 3 (11 = 5 + 5 + 1)

Example 2: coins = [2], amount = 3 return -1.

Note: You may assume that you have an infinite number of each kind of coin.

题目大意：你有一些不同面值的硬币，你需要用这些硬币凑成amount值，计算需要的最少的硬币个数～如果不能凑成，就返回-1

分析：设立dp数组（dp[i]表示凑成i需要的最小硬币个数），一开始dp[0]等于0，i从1到amount填充dp[i]的值，第二层循环遍历coins数组，假设coins[j]作为第一个硬币，那么dp[i-coins[j]]表示剩下的金额需要的硬币个数，此时判断i-coins[j]是否等于-1，也就是是否能够凑成，如果能够凑成，那么当dp[i]>0时表示凑成i需要dp[i]个硬币，这种情况下如果dp[i-coins[j]]+1更小，就更新dp[i]，否则就保留dp[i]的值。如果dp[i]本身就<=0说明前面没有方法可以凑成i，那么就直接更新dp[i]为dp[i-coins[j]] + 1的值，有一种凑成的方法总比没有好～最后返回dp[amount]的值表示凑成amount数量需要的最小硬币个数，如果不存在，本身dp数组的初值就为-1，所以也会按照要求返回-1～

```
class Solution {
public:
    int coinChange(vector<int>& coins, int amount) {
        vector<int> dp(amount + 1, -1);
        dp[0] = 0;
        for (int i = 1; i <= amount; i++) {
            for (int j = 0; j < coins.size(); j++) {
                if (i >= coins[j] && dp[i-coins[j]] != -1) {
                    if (dp[i] > 0)
                        dp[i] = min(dp[i], dp[i-coins[j]] + 1);
                    else
                        dp[i] = dp[i-coins[j]] + 1;
                }
            }
        }
        return dp[amount];
    }
};
```

324. Wiggle Sort II [Medium]

Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3]....

Example: (1) Given nums = [1, 5, 1, 1, 6, 4], one possible answer is [1, 4, 1, 5, 1, 6]. (2) Given nums = [1, 3, 2, 2, 3, 1], one possible answer is [2, 3, 1, 3, 1, 2].

Note: You may assume all input has valid answer.

Follow Up: Can you do it in O(n) time and/or in-place with O(1) extra space?

题目大意：给一个数组，按照nums[0] < nums[1] > nums[2] < nums[3]...的顺序重排序数组～

分析：先给数组排序，然后拷贝一份一样的数组arr。然后p指针从数组中间开始向前，q指针从数组最后一位开始向前，依次赋值给nums[i]和nums[j]——其中i是偶数下标，j是奇数下标～

```cpp
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        vector<int> arr = nums;
        int p = (nums.size() - 1) / 2, q = nums.size() - 1;
        for (int i = 0, j = 1; i < nums.size(); i += 2, j += 2) {
            nums[i] = arr[p--];
            if (j < nums.size()) nums[j] = arr[q--];
        }
    }
};
```

326. Power of Three [Easy]

Given an integer, write a function to determine if it is a power of three.

Follow up: Could you do it without using any loop / recursion?

```cpp
class Solution {
public:
    bool isPowerOfThree(int n) {
        if(n <= 0)
            return false;
        return pow(3, (round)(log(n) / log(3))) == n;
    }
};
```

328. Odd Even Linked List [Medium]

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example: Given 1->2->3->4->5->NULL, return 1->3->5->2->4->NULL.

Note: The relative order inside both the even and odd groups should remain as it was in the input. The first node is considered odd, the second node even and so on ...

题目大意：给一个单链表，将奇数结点放到一起，然后将偶数结点放到一起并放在奇数结点的后面返回～

分析：设立odd、even和evenHead三个指针，odd负责将所有奇数结点连接在一起，even负责把所有偶数结点连接在一起，而evenHead指向head->next表示偶数结点的头结点。将所有的串联好后将odd->next = evenHead, 最后返回head～

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if (head == NULL) return head;
        ListNode *odd = head, *even = head->next, *evenHead = head->next;
        while (even != NULL && even->next != NULL) {
            odd->next = odd->next->next;
            even->next = even->next->next;
            odd = odd->next;
            even = even->next;
        }
        odd->next = evenHead;
        return head;
    }
};
```

330. Patching Array [Hard]

Given a sorted positive integer array nums and an integer n, add/patch elements to the array such that any number in range [1, n] inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

Example 1: nums = [1, 3], n = 6 Return 1.

Combinations of nums are [1], [3], [1,3], which form possible sums of: 1, 3, 4. Now if we add/patch 2 to nums, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3]. Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6]. So we only need 1 patch.

Example 2: nums = [1, 5, 10], n = 20 Return 2. The two patches can be [2, 4].

Example 3: nums = [1, 2, 2], n = 5 Return 0.

题目大意： 给一个有序非负整数列nums和一个整数n，最少需要添加多少个数可以使得[1,n]间的每一个数都可以被数列中若干个数的和来表示。输出最小需要添加多少个数字。

分析： 假设[0,t)是暂时能够满足题意的区间，当t<=n的时候，对于每下一个nums[i]： 0.如果nums[i]比t小，那么[0,t)区间内的每一个数加上nums[i]后，区间就被扩展为[0,t+nums[i])了。 1.如果nums[i]比t大，那么必须要加入一个数才能满足扩展区间后的连续性。能加入的这个数当然是越大越好，但是不能超过t，因为超过t的话就会有t之后这个数之前的区间不满足。所以可以令t=t+t，这是它能扩展的最大区间了。此时用cnt计数，cnt+1表示加入了一个数（这个数是t）。 注意点：一开始全都用的int结果在测试用例Last executed input:[1,2,31,33] 2147483647 这个用例下面超时了 后来机智的想了一下发现是如果t当前的值很大又执行了t = t + t会溢出导致t变成了一个很小的值，然后再while肯定超时啦。 所以把int t = 1改为long long int类型就AC啦。啦啦啦。

```
class Solution {
public:
```

```
    int minPatches(vector<int>& nums, int n) {
        int len = nums.size();
        long long int t = 1;
        int cnt = 0;
        int i = 0;
        while(t <= n) {
            if(i < len && nums[i] <= t) {
                t = t + nums[i];
                i++;
            } else {
                t = t + t;
                cnt++;
            }
        }
        return cnt;
    }
};
```

338. Counting Bits [Medium]

Given a non negative integer number num. For every numbers i in the range 0 ≤ i ≤ num calculate the number of 1's in their binary representation and return them as an array.

Example: For num = 5 you should return [0,1,1,2,1,2].

Follow up:

It is very easy to come up with a solution with run time O(n*sizeof(integer)). But can you do it in linear time O(n) /possibly in a single pass? Space complexity should be O(n). Can you do it like a boss? Do it without using any builtin function like __builtin_popcount in c++ or in any other language.

```
update v3.0:
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> v(num + 1);
        v[0] = 0;
        for(int i = 1; i < num + 1; i++) {
            v[i] = v[i >> 1] + i % 2;
        }
        return v;
    }
};


update v2.0:
class Solution {
```

```
public:
    vector<int> countBits(int num) {
        vector<int> v(num + 1);
        v[0] = 0;
        v[1] = 1;
        v[2] = 1;
        v[3] = 2;
        for(int i = 4; i < num + 1; i = i * 2) {
            for(int j = i; j < i + i/2 && j < num + 1; j++) {
                v[j] = v[j - i/2];
            }
            for(int j = i + i/2; j < i * 2 && j < num + 1; j++) {
                int temp = j;
                int cnt = 0;
                while(temp) {
                    cnt = cnt + temp % 2;
                    temp = temp/2;
                }
                v[j] = cnt;
            }
        }
        return v;
    }
};


version v1.0:
class Solution {
public:
    vector<int> countBits(int num) {
        vector<int> v(num + 1);
        for(int i = 0; i <= num; i++) {
            int temp = i;
            int cnt = 0;
            while(temp) {
                cnt = cnt + temp % 2;
                temp = temp / 2;
            }
            v[i] = cnt;
        }
        return v;
    }
};
```

343. Integer Break [Medium]

Given a positive integer n, break it into the sum of at least two positive integers and maximize the product of those integers. Return the maximum product you can get.

For example, given n = 2, return 1 (2 = 1 + 1); given n = 10, return 36 (10 = 3 + 3 + 4).

Note: you may assume that n is not less than 2.

Hint:

There is a simple O(n) solution to this problem. You may check the breaking results of n ranging from 7 to 10 to discover the regularities.

我们知道...某数的n次方结果总是比乘法大... 如果可以拆成3...那就不要选择2...包括4(是2的平方) 如果可以拆成5...依旧小于3×2=6 如果可以拆成6...依旧小于3×3=9 如果可以拆成7...依旧小于3x2x2=9 如果可以拆成8...依旧小于3x3x2=18 如果可以拆成9...依旧小于3*3*3=27 所以说拆成大数还不如变成3的次方形式。。 4 = 2 + 2 5 = 2 + 3 6 = 3 + 3 7 = 3 + 3 + 1 = 3 + 4 8 = 3 + 3 + 2 9 = 3 + 3 + 3 10 = 3 + 3 + 3 + 1 = 3 + 3 + 4 11 = 3 + 3 + 3 + 2 12 = 3 + 3 + 3 + 3 我的意思是... 如果有3 + 1 出现，那它是小于2的2次方的... 所以如果余数是1，就把3×1改成x4...

如果有3 + 2出现，那就没法拆成次方形式... 所以如果余数是2，依然是3×2...

如果有3 + 3出现，那一定大于 2的3次方... 所以如果余数是0，那就是3×3...

就是说... 尽可能拆成3的n次方形式...如果余数是1那就把3*1变成2*2... 找到规律后，代码就很简单，如下：

```
class Solution {
public:
    int integerBreak(int n) {
        if(n <= 3)
            return n - 1;
        int cnt = 1;
        while(n > 2) {
            cnt = cnt * 3;
            n = n - 3;
        }
        if(n == 0)
            return cnt;
        else if(n == 1)
            return cnt / 3 * 4;
        else
            return cnt * 2;
    }
};
```

344. Reverse String [Easy]

Write a function that takes a string as input and returns the string reversed.

Example: Given s = “hello”, return “olleh”.

```
//方法一：最简单的办法，直接调用 reverse 函数。。
class Solution {
```

```
public:
    string reverseString(string s) {
        reverse(s.begin(), s.end());
        return s;
    }
};

//方法二：比较常规的做法，左右两边第一个和倒数第一个调换，第二个和倒数第二个调换...
class Solution {
public:
    string reverseString(string s) {
        for(int i = 0; i < s.length() / 2; i++) {
            swap(s[i], s[s.length() - i - 1]);
        }
        return s;
    }
};
```

345. Reverse Vowels of a String [Easy]

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1: Given s = “hello”, return “holle”.

Example 2: Given s = “leetcode”, return “leotcede”.

```
class Solution {
public:
    string reverseVowels(string s) {
        int i = 0, j = s.length() - 1;
        while(i < j) {
            while(i < j && s[i] != 'a' && s[i] != 'e' && s[i] != 'i' && s[i] != 'o' && s[i] != 'u'
                        && s[i] != 'A' && s[i] != 'E' && s[i] != 'I' && s[i] != 'O' && s[i] != 'U') {
                i++;
            }
            while(i < j && s[j] != 'a' && s[j] != 'e' && s[j] != 'i' && s[j] != 'o' && s[j] != 'u'
                        && s[j] != 'A' && s[j] != 'E' && s[j] != 'I' && s[j] != 'O' && s[j] != 'U') {
                j--;
            }
            if(i < j) {
                swap(s[i], s[j]);
            }
            i++;
            j--;
```

```
        }
        return s;
    }
};
```

347. Top K Frequent Elements [Medium]

Given a non-empty array of integers, return the k most frequent elements. For example, Given [1,1,1,2,2,3] and k = 2, return [1,2]. Note: You may assume k is always valid, 1 ≤ k ≤ number of unique elements. Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

分析：先遍历数组将数组的值与出现的次数用map映射，因为根据map的值来排序，所以用pair<int, int>类型的vector转存后用sort实现排序方法。从大到小排序，然后取前k个值储存在新的vector v中返回v。

```
typedef pair<int, int> PAIR;
class Solution {
public:
    static int cmp(const PAIR &x, const PAIR &y) {
        return x.second > y.second;
    }
    vector<int> topKFrequent(vector<int>& nums, int k) {
        vector<int> v;
        map<int, int> m;
        vector<PAIR> pair_vec;
        for(int i = 0; i < nums.size(); i++) {
            m[nums[i]]++;
        }
        for(map<int, int>::iterator it = m.begin(); it != m.end();
it++) {
            pair_vec.push_back(make_pair(it->first, it->second));
        }
        sort(pair_vec.begin(), pair_vec.end(), cmp);
        vector<PAIR>::iterator curr = pair_vec.begin();
        while(k--) {
            v.push_back(curr->first);
            curr++;
        }
        return v;
    }
};
```

349. Intersection of Two Arrays [Easy]

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2].

Note: Each element in the result must be unique. The result can be in any order.

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        set<int> s1;
        set<int> s2;
        for(int i = 0; i < nums1.size(); i++) {
            s1.insert(nums1[i]);
        }
        for(int i = 0; i < nums2.size(); i++) {
            if(s1.find(nums2[i]) != s1.end()) {
                s2.insert(nums2[i]);
            }
        }
        vector<int> v;
        for(set<int>::iterator it = s2.begin(); it != s2.end(); it++) {
            v.push_back(*it);
        }
        return v;
    }
};
```

350. Intersection of Two Arrays II [Easy]

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].

Note: Each element in the result should appear as many times as it shows in both arrays. The result can be in any order.

分析：将nums1的每一个数字以及对应数字的个数存储在map里面，遍历nums2中的所有元素，如果当前元素在map中存在，就把它放入result数组中，并将其数量-1。最终返回result即为所求交集～

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        vector<int> result;
        map<int, int> m;
        for(int i = 0; i < nums1.size(); i++)
            m[nums1[i]]++;
        for(int i = 0; i < nums2.size(); i++) {
            if(m[nums2[i]] != 0) {
                m[nums2[i]]--;
                result.push_back(nums2[i]);
            }
```

```
        }
        return result;
    }
};
```

357. Count Numbers with Unique Digits [Medium]

Given a non-negative integer n, count all numbers with unique digits, x, where 0 ≤ x < 10n.

Example: Given n = 2, return 91. (The answer should be the total numbers in the range of 0 ≤ x < 100, excluding [11,22,33,44,55,66,77,88,99])

Hint:

A direct way is to use the backtracking approach. Backtracking should contains three states which are (the current number, number of steps to get that number and a bitmask which represent which number is marked as visited so far in the current number). Start with state (0,0,0) and count all valid number till we reach number of steps equals to 10n. This problem can also be solved using a dynamic programming approach and some knowledge of combinatorics. Let f(k) = count of numbers with unique digits with length equals k. f(1) = 10, ..., f(k) = 9 * 9 * 8 * ... (9 – k + 2) [The first factor is 9 because a number cannot start with 0].

题目大意：给一个数n，求0～n位数间所有数字中，每一位数字都各不相同的数字的个数～

分析：根据提示中所给的公式，1位数字有10个，第k位有f(k) = 9 * 9 * 8 * ... (9 – k + 2)个，累加从2位到n位的f(k)的总和，再加上1位的10个数字，即为所求～

```
class Solution {
public:
    int countNumbersWithUniqueDigits(int n) {
        if(n == 0) return 1;
        int result = 10, cnt = 9;
        for(int i = 2; i <= n; i++) {
            cnt *= (11 - i);
            result += cnt;
        }
        return result;
    }
};
```

365. Water and Jug Problem [Medium]

You are given two jugs with capacities x and y litres. There is an infinite amount of water supply available. You need to determine whether it is possible to measure exactly z litres using these two jugs.

If z liters of water is measurable, you must have z liters of water contained within one or both buckets by the end.

Operations allowed:

Fill any of the jugs completely with water. Empty any of the jugs. Pour water from one jug into another till the other jug is completely full or the first jug itself is empty. Example 1: (From the famous “Die Hard” example)

Input: x = 3, y = 5, z = 4 Output: True Example 2:

Input: x = 2, y = 6, z = 5 Output: False

题目大意：给两个容量为x和y升的容器，和无限多的水。你需要确定是否可能用x和y量出z升的水，z升的水必须用一个或者两个容器盛装～

分析：首先x和y的总容量一定要大于z。其次如果z == 0则直接return true; 最后需要满足z必须是x和y的最大公约数的倍数～最大公约数采用辗转相除法求得～

```
class Solution {
public:
    bool canMeasureWater(int x, int y, int z) {
        return z == 0 || (x + y >= z && z % gcd(x, y) == 0);
    }
private:
    int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
};
```

367. Valid Perfect Square [Medium]

Given a positive integer num, write a function which returns True if num is a perfect square else False.

Note: Do not use any built-in library function such as sqrt.

Example 1:

Input: 16 Returns: True Example 2:

Input: 14 Returns: False

题目大意：判断所给的数是否正好是某数的平方～

分析：二分查找法，假设它是某数的平方，将该数的结果控制在left和right之间，不断循环，每次令mid为left和right的中间值，如果mid的平方等于num说明返回true；如果mid的平方小于num并且mid+1的平方大于num，说明num不是任何数的平方，夹在mid和mid+1之间～如果mid的平方小于num说明结果在mid的右边，令left = mid + 1；否则某数就是在mid的左边，right = mid – 1～～

```
class Solution {
public:
    bool isPerfectSquare(int num) {
        long left = 0, right = INT_MAX, mid = 0;
```

```
        while (true) {
            mid = left + (right - left) / 2;
            if (mid * mid == num)
                return true;
            if (mid * mid < num && (mid + 1) * (mid + 1) > num)
                return false;
            if (mid * mid < num)
                left = mid + 1;
            else
                right = mid - 1;
        }
    }
};
```

371. Sum of Two Integers [Easy]

Calculate the sum of two integers a and b, but you are not allowed to use the operator + and -.

Example: Given a = 1 and b = 2, return 3.

分析：位运算~ 首先，已知异或（就是这个“^”符号）可以得到： 0^0 = 0 0^1 = 1 1^1 = 0 正是位相加时该位的结果~（只不过还有个进位没加罢了~） 所以对于还没有加进位的result，result可以暂时等于a^b

其次，已知与运算（就是这个“&”符号）可以得到： 0&0 = 0 0&1 = 0 1&1 = 1 正是位相加时候有进位的那一位标注为了1~ 但是进位是往前一个位相加上去的呀~ 所以carry = (a & b) << 1

现在处理要把result加上进位的事情～ 如果进位carry等于0，那么不用加～直接等于result的值就好了～ 如果进位不等于0，那么就要把result和carry的值按位相加～ 按位相加的结果也可能导致进位~所以先用个临时变量temp把carry的值保存，然后令carry = (result & temp) << 1（也就是result和原来carry按位相加后进位的结果～），然后result = result ^ temp（也就是result和原来carry按位相加的结果～），不断循环往复，直到有一次carry等于0，不再需要进位了～～

```
class Solution {
public:
    int getSum(int a, int b) {
        int carry = (a & b) << 1;
        int result = (a ^ b);
        while(carry != 0) {
            int temp = carry;
            carry = (result & temp) << 1;
            result = result ^ temp;
        };
        return result;
    }
};
```

374. Guess Number Higher or Lower [Easy]

We are playing the Guess Game. The game is as follows:

I pick a number from 1 to n. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API guess(int num) which returns 3 possible results (-1, 1, or 0):

-1 : My number is lower 1 : My number is higher 0 : Congrats! You got it! Example: n = 10, I pick 6.

Return 6.

分析：二分查找法解决，不过要注意mid = low + (high – low) / 2 不要写成 mid = (low + high) / 2，后者超过了Int类型的最大值，相加变成负数，会导致超时～～

```
// Forward declaration of guess API.
// @param num, your guess
// @return -1 if my number is lower, 1 if my number is higher,
otherwise return 0
int guess(int num);

class Solution {
public:
    int guessNumber(int n) {
        int low = 1, high = n;
        while(low <= high) {
            int mid = low + (high - low) / 2;
            int temp = guess(mid);
            if(temp == 1)
                low = mid + 1;
            else if(temp == -1)
                high = mid - 1;
            else
                return mid;
        }
    }
};
```

377. Combination Sum IV [Medium]

Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

Example:

nums = [1, 2, 3] target = 4

The possible combination ways are: (1, 1, 1, 1) (1, 1, 2) (1, 2, 1) (1, 3) (2, 1, 1) (2, 2) (3, 1)

Note that different sequences are counted as different combinations.

Therefore the output is 7. Follow up: What if negative numbers are allowed in the given array? How does it change the problem? What limitation we need to add to the question to allow negative numbers?

题目大意：给一个数组，都是正整数且没有重复的数字，任取里面的数字，问有多少种相加等于target的方式 分析：建立dp数组，（dp[i]表示相加等于i的方法有dp[i]个）设dp[0]等于1。i从1到target，给dp[i]赋值。要想知道dp[i]的值，用j遍历nums数组，只要将dp[i-nums[j]]的值累加即可得到，注意只累加i大于等于nums[j]的情况，表示让nums[j]作为第一个数字，剩下的数字有的组合方法有dp[i-nums[j]]个，最终返回dp[target]的值即可～

```
class Solution {
public:
    int combinationSum4(vector<int>& nums, int target) {
        vector<int> dp(target + 1);
        dp[0] = 1;
        for (int i = 1; i <= target; i++)
            for (int j = 0; j < nums.size(); j++)
                if (i >= nums[j]) dp[i] += dp[i - nums[j]];
        return dp[target];
    }
};
```

378. Kth Smallest Element in a Sorted Matrix [Medium]

Given a n x n matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:

matrix = [[1, 5, 9], [10, 11, 13], [12, 13, 15]], k = 8,

return 13. Note: You may assume k is always valid, 1 ≤ k ≤ n2.

分析：使用multiset，因为是求第k小的，而且允许相同的元素，所以使用可以存储相同元素的multiset。当multiset里面的元素大于k个的时候，删除集合中最大的那个（即最后一个元素，也就是s.end()的前一个），最后输出最后一个元素即可～即 *s.rbegin()～

```
class Solution {
public:
    int kthSmallest(vector<vector<int>>& matrix, int k) {
        multiset<int> s;
        for(int i = 0; i < matrix.size(); i++) {
            for(int j = 0; j < matrix[i].size(); j++) {
                s.insert(matrix[i][j]);
                if(s.size() > k)
```

```
                    s.erase(--s.end());
                }
            }
            return *s.rbegin();
        }
};
```

382. Linked List Random Node [Medium]

Given a singly linked list, return a random node's value from the linked list. Each node must have the same probability of being chosen.

Follow up: What if the linked list is extremely large and its length is unknown to you? Could you solve this efficiently without using extra space?

Example:

// Init a singly linked list [1,2,3]. ListNode head = new ListNode(1); head.next = new ListNode(2); head.next.next = new ListNode(3); Solution solution = new Solution(head);

// getRandom() should return either 1, 2, or 3 randomly. Each element should have equal probability of returning. solution.getRandom();

题目大意：给一个单链表，返回一个随机结点的值，要求每一个结点都有同样的概率被选中～

分析：先求出链表的长度len，然后用rand() % len生成一个0～len-1之间的随机数cnt，然后从head开始遍历直到第cnt个结点，返回第cnt个结点的值～

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
private:
    ListNode* head;
    int len;
public:
    /** @param head The linked list's head.
        Note that the head is guaranteed to be not null, so it contains at least one node. */
    Solution(ListNode* head) {
        this->head = head;
        len = 1;
        while(head->next != NULL) {
            head = head->next;
```

```
            len++;
        }
    }

    /** Returns a random node's value. */
    int getRandom() {
        ListNode* node = head;
        int cnt = rand() % len;
        while(cnt--) {
            node = node->next;
        }
        return node->val;
    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(head);
 * int param_1 = obj.getRandom();
 */
```

383. Ransom Note [Easy]

Given an arbitrary ransom note string and another string containing letters from all the magazines, write a function that will return true if the ransom note can be constructed from the magazines ; otherwise, it will return false.

Each letter in the magazine string can only be used once in your ransom note.

Note: You may assume that both strings contain only lowercase letters.

canConstruct(“a”, “b”) -> false canConstruct(“aa”, “ab”) -> false canConstruct(“aa”, “aab”) -> true

题目大意：给两个string，判断第一个string能否由第二个string里面所含有的字母组成，第二个string里面的所有字母只能使用一次~

分析：建立一个hash数组，对第二个string遍历并记录每个字符出现的次数，然后遍历第一个string，如果有出现hash里面不存在的字符，那么return false～

```
class Solution {
public:
    bool canConstruct(string ransomNote, string magazine) {
        vector<int> hash(256);
        for(int i = 0; i < magazine.size(); i++)
            hash[magazine[i]]++;
        for(int i = 0; i < ransomNote.size(); i++)
            if(hash[ransomNote[i]])
                hash[ransomNote[i]]--;
            else
```

```
                return false;
            return true;
        }
};
```

384. Shuffle an Array [Medium]

Shuffle a set of numbers without duplicates.

Example:

// Init an array with set 1, 2, and 3. int[] nums = {1,2,3}; Solution solution = new Solution(nums);

// Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3] must equally likely to be returned. solution.shuffle();

// Resets the array back to its original configuration [1,2,3]. solution.reset();

// Returns the random shuffling of array [1,2,3]. solution.shuffle();

题目大意：实现洗牌算法。给定一个没有重复元素的数组，写shuffle和set函数，shuffle函数要求对nums数组进行洗牌然后返回，set函数要求返回原来的nums数组～ 分析：设立两个数组nums和origin，如果set就返回不会更改的origin数组，如果要洗牌，从nums的最后一位开始，设index为0～n-1中的一个随机数，产生后将index与最后一位元素进行交换，这样最后一位就算敲定了，然后再往前一位，假设当前位的下标是k，那就产生0～k之间的一个随机数，然后交换，以此类推，rand() % (i + 1)表示生成0～i中的一个随机数～直到第一位为止～

```
class Solution {
private:
    vector<int> nums;
    vector<int> origin;
public:
    Solution(vector<int> nums) {
        this->nums = nums;
        this->origin = nums;
    }

    /** Resets the array to its original configuration and return it.
*/
    vector<int> reset() {
        return origin;
    }

    /** Returns a random shuffling of the array. */
    vector<int> shuffle() {
        int n = nums.size();
        for(int i = n - 1; i >= 0; i--) {
            int index = rand() % (i + 1);
            swap(nums[i], nums[index]);
```

```
        }
        return nums;
    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(nums);
 * vector<int> param_1 = obj.reset();
 * vector<int> param_2 = obj.shuffle();
 */
```

387. First Unique Character in a String [Easy]

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

s = "leetcode" return 0.

s = "loveleetcode", return 2. Note: You may assume the string contain only lowercase letters.

分析：map存储每个字符的出现次数，从头遍历数组return第一个m[s[i]] == 1的下标。如果没有就return -1。

```
class Solution {
public:
    int firstUniqChar(string s) {
        map<char, int> m;
        for(int i = 0; i < s.length(); i++)
            m[s[i]]++;
        for(int i = 0; i < s.length(); i++) {
            if(m[s[i]] == 1)
                return i;
        }
        return -1;
    }
};
```

389. Find the Difference [Easy]

Given two strings s and t which consist of only lowercase letters.

String t is generated by random shuffling string s and then add one more letter at a random position.

Find the letter that was added in t.

Example:

Input: s = “abcd” t = “abcde”

Output: e

Explanation: ‘e’ is the letter that was added.

分析：将字符串s和字符串t的字符个数标记在hash1和hash2数组中，然后比较hash1和hash2，值不同的那个字符即为所求～

```
class Solution {
public:
    char findTheDifference(string s, string t) {
        int hash1[256] = {0}, hash2[256] = {0};
        for(int i = 0; i < s.length(); i++)
            hash1[s[i]]++;
        for(int i = 0; i < t.length(); i++)
            hash2[t[i]]++;
        for(int i = 0; i < 256; i++)
            if(hash1[i] != hash2[i])
                return (char)i;
        return '0';
    }
};
```

392. Is Subsequence [Medium]

Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t is potentially a very long (length ~= 500,000) string, and s is a short string (<=100). A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, “ace” is a subsequence of “abcde” while “aec” is not). Example 1: s = “abc”, t = “ahbgdc” Return true. Example 2: s = “axc”, t = “ahbgdc” Return false. Follow up: If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you want to check one by one to see if T has its subsequence. In this scenario, how would you change your code?

题目大意：判断s是否是t的子串～

分析：设立两个指针p和q，分别指向s[0]和t[0]，只要p和q没有超出s和t的长度范围，不断判断当前s[p]与t[q]是否相等，如果不相等就将q指针不断后移，直到相等为止～相等后p和q指针同时后移，依次判断～ 最终判断p指针是否等于lens，即p指针是否指完了s字符串的所有字符～

```
class Solution {
public:
    bool isSubsequence(string s, string t) {
        int p = 0, q = 0, lens = s.length(), lent = t.length();
```

```
        while(p < lens && q < lent) {
            while(q < lent && s[p] != t[q]) q++;
            if(s[p] == t[q]) {
                p++;
                q++;
            }
        }
        return p == lens;
    }
};
```

396. Rotate Function [Easy]

Given an array of integers A and let n to be its length.

Assume Bk to be an array obtained by rotating the array A k positions clock-wise, we define a "rotation function" F on A as follow:

F(k) = 0 * Bk[0] + 1 * Bk[1] + ... + (n-1) * Bk[n-1].

Calculate the maximum value of F(0), F(1), ..., F(n-1).

Note: n is guaranteed to be less than 105.

Example:

A = [4, 3, 2, 6]

F(0) = (0 * 4) + (1 * 3) + (2 * 2) + (3 * 6) = 0 + 3 + 4 + 18 = 25 F(1) = (0 * 6) + (1 * 4) + (2 * 3) + (3 * 2) = 0 + 4 + 6 + 6 = 16 F(2) = (0 * 2) + (1 * 6) + (2 * 4) + (3 * 3) = 0 + 6 + 8 + 9 = 23 F(3) = (0 * 3) + (1 * 2) + (2 * 6) + (3 * 4) = 0 + 2 + 12 + 12 = 26

So the maximum value of F(0), F(1), F(2), F(3) is F(3) = 26.

分析：算出每个F[i]，求出最大值maxn～每个F[i]可由前一步F[i-1]得到，F[i] = F[i – 1] + sum – len * A[len – i]，其中sum为A[i]总和，len为A[i]长度～找到这个关系后就能求得F[i]的所有值～

```
class Solution {
public:
    int maxRotateFunction(vector<int>& A) {
        int len = A.size(), sum = 0;
        if(len == 0)
            return 0;
        vector<int> F(len);
        for(int i = 0; i < len; i++) {
            sum += A[i];
            F[0] += i * A[i];
        }
        int maxn = F[0];
        for(int i = 1; i < len; i++) {
            F[i] = F[i - 1] + sum - len * A[len - i];
```

```
            maxn = max(F[i], maxn);
        }
        return maxn;
    }
};
```

398. Random Pick Index [Medium]

Given an array of integers with possible duplicates, randomly output the index of a given target number. You can assume that the given target number must exist in the array.

Note: The array size can be very large. Solution that uses too much extra space will not pass the judge.

Example:

int[] nums = new int[] {1,2,3,3,3}; Solution solution = new Solution(nums);

// pick(3) should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning. solution.pick(3);

// pick(1) should return 0. Since in the array only nums[0] is equal to 1. solution.pick(1);

题目大意：给一个整数数组，里面可能有重复元素。给一个target，随机返回一个数组元素等于target的元素下标～

分析：将所有等于target的元素的下标存储在index数组中，然后根据rand() % index.size()的值随机从index中取一个值返回～

```
class Solution {
private:
    vector<int> nums;
public:
    Solution(vector<int> nums) {
        this->nums = nums;
    }

    int pick(int target) {
        vector<int> index;
        for(int i = 0; i < nums.size(); i++) {
            if(nums[i] == target)
                index.push_back(i);
        }
        return index[rand() % index.size()];
    }
};

/**
 * Your Solution object will be instantiated and called as such:
 * Solution obj = new Solution(nums);
```

```
 * int param_1 = obj.pick(target);
 */
```

400. Nth Digit [Easy]

Find the nth digit of the infinite integer sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

Note: n is positive and will fit within the range of a 32-bit signed integer (n < 231).

Example 1:

Input: 3

Output: 3 Example 2:

Input: 11

Output: 0

Explanation: The 11th digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.

分析：个位数：1-9，一共9个,共计9个数字；2位数：10-99,一共90个，共计180个数字； 3位数：100-999，一共900个，共计270个数字.....以此类推，所以先算出它在几位数的区间，然后算出它是在这个区间内的第几个数，最后算出在这个数的第几位～

```
class Solution {
public:
    int findNthDigit(int n) {
        long digit = 1, sum = 9;
        while(n > digit * sum) {
            n = n - digit * sum;
            sum = sum * 10;
            digit++;
        }
        int index = n % digit;
        if(index == 0)
            index = digit;
        long num = pow(10, digit - 1);
        num += (index == digit) ? (n / digit - 1) : (n / digit);
        for(int i = index; i < digit; i++)
            num = num / 10;
        return num % 10;
    }
};
```

401. Binary Watch [Easy]

A binary watch has 4 LEDs on the top which represent the hours (0-11), and the 6 LEDs on the bottom represent the minutes (0-59).

Each LED represents a zero or one, with the least significant bit on the right.

For example, the above binary watch reads “3:25”.

Given a non-negative integer n which represents the number of LEDs that are currently on, return all possible times the watch could represent.

Example:

Input: n = 1 Return: [“1:00”, “2:00”, “4:00”, “8:00”, “0:01”, “0:02”, “0:04”, “0:08”, “0:16”, “0:32”]

分析：小时数h从0～11，分钟数m从0～59，把他们每一个转换为一个二进制整体，初始化为bitset，看bitset中1的个数是否为num，如果是就将h和m构成一个字符串加入到result字符串数组中，最后返回result～～

```
class Solution {
public:
    vector<string> readBinaryWatch(int num) {
        vector<string> result;
        for(int h = 0; h < 12; h++) {
            for(int m = 0; m < 60; m++) {
                bitset<10> b(h << 6 | m);
                if(b.count() == num) {
                    string temp = to_string(h) + ":";
                    if(m < 10)
                        temp += "0";
                    temp += to_string(m);
                    result.push_back(temp);
                }
            }
        }
        return result;
    }
};
```

402. Remove K Digits [Medium]

Given a non-negative integer num represented as a string, remove k digits from the number so that the new number is the smallest possible.

Note: The length of num is less than 10002 and will be ≥ k. The given num does not contain any leading zero. Example 1:

Input: num = “1432219”, k = 3 Output: “1219” Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest. Example 2:

Input: num = “10200”, k = 1 Output: “200” Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes. Example 3:

Input: num = “10”, k = 2 Output: “0” Explanation: Remove all the digits from the number and it is left with nothing which is 0.

题目大意：给一个以字符串形式表示的非负整数，从字符串中移除k位，使得剩下的数字尽可能的小～

分析：将字符串中每一个元素放入栈中，如果当前要放入的元素比栈顶元素大，而且k > 0（还需要移除数字），则将栈顶元素弹出后再放入新的元素，因为前面的数字越小越好～等到所有数字都加入栈中后，如果k依旧>0，也就是说还有需要弹栈的数字，那就将最后几位移除，因为前面的数字肯定比后面的数字小～将栈中所有元素放入result字符串中，然后再用index判断第一个不为0的下标为多少，然后截取result为result.substr(index)去除了前导0，如果最后发现result为空则返回”0"，否则返回result～

```
class Solution {
public:
    string removeKdigits(string num, int k) {
        string result = "";
        stack<char> s;
        for (int i = 0; i < num.size(); i++) {
            while (k > 0 && !s.empty() && s.top() > num[i]) {
                k--;
                s.pop();
            }
            s.push(num[i]);
        }
        while (k > 0 && !s.empty()) {
            k--;
            s.pop();
        }
        while (!s.empty()) {
            result = s.top() + result;
            s.pop();
        }
        int index = 0;
        while (result[index] == '0') index++;
        result = result.substr(index);
        if (result == "") return "0";
        return result;

    }
};
```

404. Sum of Left Leaves [Easy]

Find the sum of all left leaves in a given binary tree.

Example:

3 / \ 9 20 / \ 15 7

There are two left leaves in the binary tree, with values 9 and 15 respectively. Return 24.

分析：深度优先遍历，将所有结点从根结点开始遍历一遍，设立isLeft的值，当当前结点是叶子节点并且也是左边，那就result加上它的值～

```
/**
 * 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 result = 0;
    int sumOfLeftLeaves(TreeNode* root) {
        if(root == NULL)
            return 0;
        dfs(root, false);
        return result;
    }
    void dfs(TreeNode* root, bool isLeft) {
        if(root->left == NULL && root->right == NULL) {
            if(isLeft == true)
                result += root->val;
            return ;
        }
        if(root->left != NULL)
            dfs(root->left, true);
        if(root->right != NULL)
            dfs(root->right, false);
    }
};
```

405. Convert a Number to Hexadecimal [Easy]

Given an integer, write an algorithm to convert it to hexadecimal. For negative integer, two's complement method is used.

Note:

All letters in hexadecimal (a-f) must be in lowercase. The hexadecimal string must not contain extra leading 0s. If the number is zero, it is represented by a single zero character '0'; otherwise, the first character in the hexadecimal string will not be the zero character. The given number is guaranteed to fit within the range of a 32-bit signed integer. You must not use any method provided by the library which converts/formats the number to hex directly. Example 1:

Input: 26

Output: “1a” Example 2:

Input: -1

Output: “ffffffff”

分析：就按照十进制转十六进制的正常逻辑来计算，先把十进制转换成二进制，然后二进制每4位转换成一个十六进制字符。对于十进制转二进制，只需要将num与15取&，并将num右移四位即可，因为不超过32位二进制，所以只需要8次即可～然后因为这样得到的result字符串是倒过来的，所以需要reverse一下～此时可能前面有多余的0，去除多余的前导0即可得到所求的十六进制result字符串～

```
class Solution {
public:
    string toHex(int num) {
        string result = "", s = "0123456789abcdef";
        for(int i = 1; i <= 8; i++) {
            result += s[num & 15];
            num = num >> 4;
        }
        reverse(result.begin(), result.end());
        while(result.length() > 1 && result[0] == '0')
            result = result.substr(1);
        return result;
    }
};
```

406. Queue Reconstruction by Height [Medium]

Suppose you have a random list of people standing in a queue. Each person is described by a pair of integers (h, k), where h is the height of the person and k is the number of people in front of this person who have a height greater than or equal to h. Write an algorithm to reconstruct the queue.

Note: The number of people is less than 1,100.

Example

Input: [[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

Output: [[5,0], [7,0], [5,2], [6,1], [4,4], [7,1]]

题目大意：给一组序列(h, k), h表示身高，k表示在当前人的前面有k个人比他高，求满足这样的排队序列～

分析：先将所有人按照身高h排序，如果他们的身高h相同就按照k从小到大排序～接下来就可以构建result数组了，从排序好的序列第一个数开始一直遍历到最后一个，依次把它插入到result数组的第k位，因为对于当前这个人，他说了比他高的人是k个，而整个数组按照身高排序了，所以前面的所有人都是比他高的，排他的队伍的时候只需要排到第k位就能满足他前面比他高的人是k个～这样将所有人都插入后得到的result即为所求队伍的顺序～～

```cpp
class Solution {
public:
    vector<pair<int, int>> reconstructQueue(vector<pair<int, int>>&
people) {
        vector<pair<int, int>> result;
        if(people.size() == 0)
            return result;
        auto cmp = [](pair<int, int> p1, pair<int, int> p2) {
            if(p1.first == p2.first)
                return p1.second < p2.second;
            else
                return p1.first > p2.first;
        };
        sort(people.begin(), people.end(), cmp);
        for(auto it : people) {
            result.insert(result.begin() + it.second, it);
        }
        return result;
    }
};
```

409. Longest Palindrome [Easy]

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

This is case sensitive, for example “Aa” is not considered a palindrome here.

Note: Assume the length of given string will not exceed 1,010.

Example:

Input: “abccccdd”

Output: 7

Explanation: One longest palindrome that can be built is “dccaccd”, whose length is 7.

分析：统计每个字母出现的字数，如果是偶数就能放到回文串里；如果是奇数就只能hash[i] – 1个放到回文串里面，而且如果存在是奇数个数的字母，则最后可以加1，也就是把落单的字母中的任意一个放到回文串的最中间使长度加1～

```cpp
class Solution {
public:
    int longestPalindrome(string s) {
        int hash[256] = {0}, len = 0, flag = 0;
        for(int i = 0; i < s.length(); i++)
            hash[s[i]]++;
```

```
        for(int i = 0; i < 256; i++) {
            if(hash[i] % 2 == 0) {
                len += hash[i];
            } else {
                len += (hash[i] - 1);
                flag = 1;
            }
        }
        return len + flag;
    }
};
```

412. Fizz Buzz [Easy]

Write a program that outputs the string representation of numbers from 1 to n.

But for multiples of three it should output “Fizz” instead of the number and for the multiples of five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”.

Example:

n = 15,

Return: [“1”, “2”, “Fizz”, “4”, “Buzz”, “Fizz”, “7”, “8”, “Fizz”, “Buzz”, “11”, “Fizz”, “13”, “14”, “FizzBuzz”]

分析：几个if else 语句分别对是否能被15整除、5整除、3整除进行赋值～

```
class Solution {
public:
    vector<string> fizzBuzz(int n) {
        vector<string> result(n);
        for(int i = 0; i < n; i++) {
            if((i + 1) % 15 == 0)
                result[i] = "FizzBuzz";
            else if((i + 1) % 5 == 0)
                result[i] = "Buzz";
            else if((i + 1) % 3 == 0)
                result[i] = "Fizz";
            else
                result[i] = to_string(i + 1);
        }
        return result;
    }
};
```

413. Arithmetic Slices [Medium]

A sequence of number is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequence:

1, 3, 5, 7, 9 7, 7, 7, 7 3, -1, -5, -9 The following sequence is not arithmetic.

1, 1, 2, 5, 7

A zero-indexed array A consisting of N numbers is given. A slice of that array is any pair of integers (P, Q) such that 0 <= P < Q < N.

A slice (P, Q) of array A is called arithmetic if the sequence: A[P], A[p + 1], ..., A[Q – 1], A[Q] is arithmetic. In particular, this means that P + 1 < Q.

The function should return the number of arithmetic slices in the array A. Example:

A = [1, 2, 3, 4]

return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itself.

题目大意：求一个数组中能构成至少连续三个数的等差数列的个数

分析：有一个规律是： 连续3个数构成等差数列，能组成的等差数列个数是1 连续4个数构成等差数列，能组成的等差数列个数是3（1+2） 连续5个数构成等差数列，能组成的等差数列个数是6（1+2+3） ..... 所以每次当有三个构成等差数列的时候，cnt = 1，之后每连续增加一个数满足与之前的数构成等差数列，就令cnt++，并且将结果加到最终的result里面～

```
class Solution {
public:
    int numberOfArithmeticSlices(vector<int>& A) {
        int cnt = 0, result = 0;
        for(int i = 2; i < A.size(); i++) {
            if(A[i-1] - A[i-2] == A[i] - A[i-1]) {
                cnt++;
                result += cnt;
            } else {
                cnt = 0;
            }
        }
        return result;
    }
};
```

415. Add Strings [Easy]

Given two non-negative integers num1 and num2 represented as string, return the sum of num1 and num2.

Note:

The length of both num1 and num2 is < 5100. Both num1 and num2 contains only digits 0-9. Both num1 and num2 does not contain any leading zero. You must not use any built-in BigInteger library or convert the inputs to integer directly.

分析：先将num1和num2倒置，然后从头到尾依次加，记得标记是否有进位sign，每次也要把进位的值加进去～如果num1加完了num2还没加完，就继续加num2的～反之同理～最后sign也要考虑有没有最高位1～所有都处理完后把result倒置～

```
class Solution {
public:
    string addStrings(string num1, string num2) {
        int len1 = num1.length(), len2 = num2.length();
        for(int i = 0; i < len1 / 2; i++)
            swap(num1[i], num1[len1 - i - 1]);
        for(int i = 0; i < len2 / 2; i++)
            swap(num2[i], num2[len2 - i - 1]);
        string result = "";
        int sign = 0, p = 0, q = 0;
        while(p < len1 && q < len2) {
            int t = (num1[p] - '0') + (num2[q] - '0') + sign;
            if(t >= 10) {
                t = t - 10;
                sign = 1;
            } else {
                sign = 0;
            }
            result += (char)(t + '0');
            p++;
            q++;
        }
        while(p < len1) {
            int t = (num1[p] - '0') + sign;
            if(t >= 10) {
                t = t - 10;
                sign = 1;
            } else {
                sign = 0;
            }
            result += (char)(t + '0');
            p++;
        }
        while(q < len2) {
            int t = (num2[q] - '0') + sign;
            if(t >= 10) {
                t = t - 10;
                sign = 1;
            } else {
                sign = 0;
            }
            result += (char)(t + '0');
            q++;
```

```
        }
        if(sign == 1)
            result += '1';
        int len = result.length();
        for(int i = 0; i < len / 2; i++) {
            swap(result[i], result[len - 1 - i]);
        }
        return result;
    }
};
```

419. Battleships in a Board [Medium]

Given an 2D board, count how many different battleships are in it. The battleships are represented with 'X's, empty slots are represented with '.'s. You may assume the following rules:

You receive a valid board, made of only battleships or empty slots. Battleships can only be placed horizontally or vertically. In other words, they can only be made of the shape 1xN (1 row, N columns) or Nx1 (N rows, 1 column), where N can be of any size. At least one horizontal or vertical cell separates between two battleships – there are no adjacent battleships. Example: X..X ...X ...X In the above board there are 2 battleships. Invalid Example: ...X XXXX ...X This is an invalid board that you will not receive – as battleships will always have a cell separating between them.

题目大意：给一张地图，战列舰用X表示，空地用.表示，战列舰只可能横着或者竖着一条，而且两条战列舰之间肯定有空地，计算战列舰的个数～

分析：其实只需要数战列舰的头部有几个就行了，因为战列舰要么横着要么竖着，它的头部肯定满足以下条件：1.它的上方是空地（或者边界） 2.它的左方是空地（或者边界） 这样遍历一遍地图上所有的点，如果当前点是X而且满足上面所述的两个条件，就将战列舰个数+1～～～

```
class Solution {
public:
    int countBattleships(vector<vector<char>>& board) {
        int result = 0;
        if(board.size() == 0) return 0;
        for(int i = 0; i < board.size(); i++) {
            for(int j = 0; j < board[0].size(); j++) {
                if(board[i][j] == 'X' && (i == 0 || board[i-1][j] == '.') && (j == 0 || board[i][j-1] == '.'))
                    result++;
            }
        }
        return result;
    }
};
```

423. Reconstruct Original Digits from English [Medium]

Given a non-empty string containing an out-of-order English representation of digits 0-9, output the digits in ascending order.

Note: Input contains only lowercase English letters. Input is guaranteed to be valid and can be transformed to its original digits. That means invalid inputs such as “abc” or “zerone” are not permitted. Input length is less than 50,000. Example 1: Input: “owoztneoer” Output: “012”

Example 2: Input: “fviefuro” Output: “45”

题目大意：给一个非空字符串，是一些数字的单词打乱后的顺序，求还原这些数字是哪些以及多少个，按照数字增序排列～

分析：先把0～9这10个数字的字母写出来，就可以知道z、x、w、g、u可以唯一确定0、6、2、8、4的个数，然后按照06284的拼写在别的字母中减去他们的拼写，接下来可以发现按照s、r、o、f、e这样的顺序分别筛出7、3、1、5、9的个数即可～（还有更简洁的代码写法，这样写是为了清晰思路）

```
class Solution {
public:
    string originalDigits(string s) {
        map<char, int> m;
        int cnt[10] = {0}, temp;
        for(int i = 0; i < s.length(); i++)
            m[s[i]]++;
        temp = m['z'];
        cnt[0] = temp;
        m['z'] -= temp;
        m['e'] -= temp;
        m['r'] -= temp;
        m['o'] -= temp;
        temp = m['x'];
        cnt[6] = temp;
        m['s'] -= temp;
        m['i'] -= temp;
        m['x'] -= temp;
        temp = m['w'];
        cnt[2] = temp;
        m['t'] -= temp;
        m['w'] -= temp;
        m['o'] -= temp;
        temp = m['g'];
        cnt[8] = temp;
        m['e'] -= temp;
        m['i'] -= temp;
        m['g'] -= temp;
        m['h'] -= temp;
        m['t'] -= temp;
        temp = m['u'];
        cnt[4] = temp;
        m['f'] -= temp;
```

```
        m['o'] -= temp;
        m['u'] -= temp;
        m['r'] -= temp;
        temp = m['s'];
        cnt[7] = temp;
        m['s'] -= temp;
        m['e'] -= temp;
        m['v'] -= temp;
        m['e'] -= temp;
        m['n'] -= temp;
        temp = m['r'];
        cnt[3] = temp;
        m['t'] -= temp;
        m['h'] -= temp;
        m['r'] -= temp;
        m['e'] -= temp;
        m['e'] -= temp;
        temp = m['o'];
        cnt[1] = temp;
        m['o'] -= temp;
        m['n'] -= temp;
        m['e'] -= temp;
        temp = m['f'];
        cnt[5] = temp;
        m['f'] -= temp;
        m['i'] -= temp;
        m['v'] -= temp;
        m['e'] -= temp;
        temp = m['e'];
        cnt[9] = temp;
        m['n'] -= temp;
        m['i'] -= temp;
        m['n'] -= temp;
        m['e'] -= temp;
        string result = "";
        for(int i = 0; i <= 9; i++)
            for(int j = 0; j < cnt[i]; j++)
                result += to_string(i);
        return result;
    }
};
```

434. Number of Segments in a String [Easy]

Count the number of segments in a string, where a segment is defined to be a contiguous sequence of non-space characters.

Please note that the string does not contain any non-printable characters.

Example:

Input: “Hello, my name is John” Output: 5

分析：一旦出现不是空格的s[i]并且他前一个不和他连续，那么就cnt++; 如果遇到是空格，重新将flag标记为0表示当前s[i]和下一个s[i]不连续～

```
class Solution {
public:
    int countSegments(string s) {
       int cnt = 0, flag = 0;
       for(int i = 0; i < s.length(); i++) {
           if(s[i] != ' ' && flag == 0) {
               cnt++;
               flag = 1;
           } else if(s[i] == ' ') {
               flag = 0;
           }
        }
        return cnt;
    }
};
```

437. Path Sum III [Easy]

You are given a binary tree in which each node contains an integer value.

Find the number of paths that sum to a given value.

The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).

The tree has no more than 1,000 nodes and the values are in the range -1,000,000 to 1,000,000.

Example:

root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

10 / \ 5 -3 / \ \ 3 2 11 / \ \ 3 -2 1

Return 3. The paths that sum to 8 are:

1. 5 -> 3
2. 5 -> 2 -> 1
3. -3 -> 11

分析：深度优先搜索，将root以及root->left、root->right结点分别作为开始结点计算下方是否有满足sum的和，dfs函数就是用来计算统计满足条件的个数的。dfs从TreeNode* root开始，查找是否满足sum和的值（此时的sum是部分和），分别dfs左边结点、右边结点，直到找到root->val == sum时候result++，在pathSum函数中返回result的值。

```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 result = 0;
    int pathSum(TreeNode* root, int sum) {
        if(root == NULL)
            return 0;
        pathSum(root->left, sum);
        pathSum(root->right, sum);
        dfs(root, sum);
        return result;
    }

    void dfs(TreeNode* root, int sum) {
        if(root == NULL) return ;
        if(root->val == sum) result++;
        dfs(root->left, sum - root->val);
        dfs(root->right, sum - root->val);
    }
};
```

438. Find All Anagrams in a String [Easy]

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.

Strings consists of lowercase English letters only and the length of both strings s and p will not be larger than 20,100.

The order of output does not matter.

Example 1:

Input: s: "cbaebabacd" p: "abc"

Output: [0, 6]

Explanation: The substring with start index = 0 is "cba", which is an anagram of "abc". The substring with start index = 6 is "bac", which is an anagram of "abc". Example 2:

Input: s: "abab" p: "ab"

Output: [0, 1, 2]

Explanation: The substring with start index = 0 is “ab”, which is an anagram of “ab”. The substring with start index = 1 is “ba”, which is an anagram of “ab”. The substring with start index = 2 is “ab”, which is an anagram of “ab”.

分析：先将p字符串的所有字母的个数标记在hash数组中。设p的字符串长度为lenp，那么从字符串s的第一位开始分别找lenp长度的字串标记temphash，比较temphash是否等于hash，如果等于说明是anagram字符串，push_back到vector里面，最后返回vector

```
class Solution {
public:
    vector<int> findAnagrams(string s, string p) {
        vector<int> result, hash(26, 0);
        int lenp = p.length(), lens = s.length();
        for(int i = 0; i < lenp; i++) {
            hash[p[i] - 'a']++;
        }
        for(int i = 0; i <= lens - lenp; i++) {
            vector<int> temphash(26, 0);
            for(int j = i; j < i + lenp; j++)
                temphash[s[j] - 'a']++;
            if(temphash == hash)
                result.push_back(i);
        }
        return result;
    }
};
```

441. Arranging Coins [Easy]

You have a total of n coins that you want to form in a staircase shape, where every k-th row must have exactly k coins.

Given n, find the total number of full staircase rows that can be formed.

n is a non-negative integer and fits within the range of a 32-bit signed integer.

Example 1: n = 5 The coins can form the following rows: ¤ ¤ ¤ ¤ ¤ Because the 3rd row is incomplete, we return 2. Example 2: n = 8 The coins can form the following rows: ¤ ¤ ¤ ¤ ¤ ¤ ¤ ¤ Because the 4th row is incomplete, we return 3.

分析：直接用公式一行代码就能解决

```
class Solution {
public:
    int arrangeCoins(int n) {
        return (int)((sqrt(8 * (long)n + 1) - 1)/ 2);
    }
};
```

442. Find All Duplicates in an Array [Medium]

Given an array of integers, 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.

Find all the elements that appear twice in this array.

Could you do it without extra space and in O(n) runtime?

Example: Input: [4,3,2,7,8,2,3,1]

Output: [2,3]

分析：题目说不能用额外的空间，就想办法直接利用nums数组标记，考虑到nums数组中的所有数都为正数，那么比如i出现过，把num[i]的值标记为负数，那么每次去看num[i]是正是负就能知道i以前有没有出现过～因为所有数都是1～n的，而下标只能0～n-1，可以把i-1标记成负数来符合这个条件～所以说可以令当前nums[i]的绝对值是num，如果nums[num-1]这个值是负数，说明以前遇到过num，那么就把num这个数字放入result数组中；如果不是负数，说明以前没有出现过这个数，就把这个数变成自己本身的负数～即：nums[num – 1] = 0 – nums[num – 1];

```
class Solution {
public:
    vector<int> findDuplicates(vector<int>& nums) {
        vector<int> result;
        for(int i = 0; i < nums.size(); i++) {
            int num = abs(nums[i]);
            if(nums[num - 1] < 0)
                result.push_back(num);
            else
                nums[num - 1] = 0 - nums[num - 1];
        }
        return result;
    }
};
```

443. String Compression [Easy]

Given an array of characters, compress it in-place.

The length after compression must always be smaller than or equal to the original array.

Every element of the array should be a character (not int) of length 1.

After you are done modifying the input array in-place, return the new length of the array.

Follow up: Could you solve it using only O(1) extra space?

Example 1: Input: [“a”,”a”,”b”,”b”,”c”,”c”,”c”]

Output: Return 6, and the first 6 characters of the input array should be: [“a”,”2",”b”,”2",”c”,”3"]

Explanation: “aa” is replaced by “a2”. “bb” is replaced by “b2”. “ccc” is replaced by “c3”. Example 2: Input: [“a”]

Output: Return 1, and the first 1 characters of the input array should be: [“a”]

Explanation: Nothing is replaced. Example 3: Input: [“a”,”b”,”b”,”b”,”b”,”b”,”b”,”b”,”b”,”b”,”b”,”b”,”b”]

Output: Return 4, and the first 4 characters of the input array should be: [“a”,”b”,”1",”2"].

Explanation: Since the character “a” does not repeat, it is not compressed. “bbbbbbbbbbbb” is replaced by “b12”. Notice each digit has its own entry in the array. Note: All characters have an ASCII value in [35, 126]. 1 <= len(chars) <= 1000.

分析：指针i指向修改内容的位置，指针j遍历整个数组chars，当下一个字符与当前字符不相同时，直接将该字符赋值到i处，然后i++，j++；否则若相同，k指向j所在位置，j继续向前出发遍历所有与k处相同的字符，则相同的个数为j-k，将j-k转化为字符串s，将s的每一个字符都赋值在i所在位置开始的chars中～最后直到j>=n时退出循环，此时i的值即为in-place后新数组中的个数～

```
class Solution {
public:
    int compress(vector<char>& chars) {
        int i = 0, j = 0, n = chars.size();
        while (j < n) {
            if (j == n - 1 || chars[j] != chars[j + 1]) {
                chars[i++] = chars[j++];
            } else {
                chars[i++] = chars[j];
                int k = j;
                while (j < n && chars[j] == chars[k]) j++;
                string s = to_string(j - k);
                for (char c : s) chars[i++] = c;
            }
        }
        return i;
    }
};
```

445. Add Two Numbers II [Medium]

You are given two non-empty linked lists representing two non-negative integers. The most significant digit comes first 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.

Follow up: What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

Example:

Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4) Output: 7 -> 8 -> 0 -> 7

题目大意：两个链表l1和l2，返回一个链表，其值为l1与l2的和～

分析：将l1和l2分别放入栈中，这样他们就被倒置了，然后依次出栈将结果相加，（不要忘记考虑进位问题），结果放入新的栈s中，然后将s弹栈，结果放入一个新链表中～～

```
/**
 * 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) {
        stack<int> s1, s2, s;
        while(l1 != NULL) {
            s1.push(l1->val);
            l1 = l1->next;
        }
        while(l2 != NULL) {
            s2.push(l2->val);
            l2 = l2->next;
        }
        int carry = 0;
        while(!s1.empty() || !s2.empty()) {
            int tempsum = carry;
            if(!s1.empty()) {
                tempsum += s1.top();
                s1.pop();
            }
            if(!s2.empty()) {
                tempsum += s2.top();
                s2.pop();
            }
            carry = 0;
            if(tempsum >= 10) {
                carry = 1;
                tempsum = tempsum - 10;
```

```
            }
            s.push(tempsum);
        }
        if(carry == 1)
            s.push(1);
        ListNode* result = new ListNode(0);
        ListNode* cur = result;
        while(!s.empty()) {
            int top = s.top();
            s.pop();
            ListNode* node = new ListNode(top);
            cur->next = node;
            cur = cur->next;
        }
        return result->next;
    }
};
```

447. Number of Boomerangs [Easy]

Given n points in the plane that are all pairwise distinct, a “boomerang” is a tuple of points (i, j, k) such that the distance between i and j equals the distance between i and k (the order of the tuple matters).

Find the number of boomerangs. You may assume that n will be at most 500 and coordinates of points are all in the range [-10000, 10000] (inclusive).

Example: Input: [[0,0],[1,0],[2,0]] Output: 2 Explanation: The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]

分析：map(m[i] = j)用来存储点与点之间距离（其实是距离的平方，不影响结果）为i的个数j。对于某一点，距离它为i的个数若为j个，则能够构成的i，j，k这样的组数是j * (j – 1)个。每次累计得到的总结果即为所求～

```
class Solution {
public:
    int numberOfBoomerangs(vector<pair<int, int>>& points) {
        int cnt = 0;
        for(int i = 0; i < points.size(); i++) {
            map<int, int> m;
            int x1 = points[i].first, y1 = points[i].second;
            for(int j = 0; j < points.size(); j++) {
                if(j == i) continue;
                int x2 = points[j].first, y2 = points[j].second;
                int dis = (x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2);
                m[dis]++;
            }
            for(auto it = m.begin(); it != m.end(); it++) {
```

```
                cnt += it->second * (it->second - 1);
            }
        }
        return cnt;
    }
};
```

448. Find All Numbers Disappeared in an Array [Easy]

Given an array of integers where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.

Find all the elements of [1, n] inclusive that do not appear in this array.

Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.

Example:

Input: [4,3,2,7,8,2,3,1]

Output: [5,6]

分析：hash[i]用来标记i是否在nums数组中出现过。然后将没有出现过的放入result数组中，返回result数组～～～

```
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        vector<int> result;
        vector<bool> hash(nums.size() + 1, false);
        for(int i = 0; i < nums.size(); i++) {
            hash[nums[i]] = true;
        }
        for(int i = 1; i < hash.size(); i++) {
            if(hash[i] == false)
                result.push_back(i);
        }
        return result;
    }
};
```

451. Sort Characters By Frequency [Medium]

Given a string, sort it in decreasing order based on the frequency of characters.

Example 1:

Input: “tree”

Output: “eert”

Explanation: ‘e’ appears twice while ‘r’ and ‘t’ both appear once. So ‘e’ must appear before both ‘r’ and ‘t’. Therefore “eetr” is also a valid answer. Example 2:

Input: “cccaaa”

Output: “cccaaa”

Explanation: Both ‘c’ and ‘a’ appear three times, so “aaaccc” is also a valid answer. Note that “cacaca” is incorrect, as the same characters must be together. Example 3:

Input: “Aabb”

Output: “bbAa”

Explanation: “bbaA” is also a valid answer, but “Aabb” is incorrect. Note that ‘A’ and ‘a’ are treated as two different characters.

分析：在cnt数组中保存每个字母出现的次数，然后按照出现次数的顺序对字符串进行排序～

```
class Solution {
public:
    string frequencySort(string s) {
        int cnt[256] = {0};
        for(int i = 0; i < s.length(); i++)
            cnt[s[i]]++;
        sort(s.begin(), s.end(), [&](char a, char b) {
            return cnt[a] > cnt[b] || (cnt[a] == cnt[b] && a < b);
        });
        return s;
    };
};
```

453. Minimum Moves to Equal Array Elements [Easy]

Contributors: amehrotra2610 Given a non-empty integer array of size n, find the minimum number of moves required to make all array elements equal, where a move is incrementing n – 1 elements by 1.

Example:

Input: [1,2,3]

Output: 3

Explanation: Only three moves are needed (remember each move increments two elements):

[1,2,3] => [2,3,3] => [3,4,3] => [4,4,4]

分析：每次n-1个数都+1，最后所有数都相等，其实等价于每次将其中一个数-1，最后所有数都相等。因为每次只能减一个数，那肯定是将除了最小数minn之外的其他所有数一次次减，直到他们等于都最小数minn。所以cnt就等于所有数与最小数之间的差距的和（因为每次只能减去一个数，且只能减去1，所以差为多少就要减去多少次～） 先求出数组的最小值minn，然后累加的所有数和minn之间的

差即为所求～

```
class Solution {
public:
    int minMoves(vector<int>& nums) {
        int minn = INT_MAX;
        for(int i = 0; i < nums.size(); i++)
            minn = min(minn, nums[i]);
        int cnt = 0;
        for(int i = 0; i < nums.size(); i++)
            cnt += nums[i] - minn;
        return cnt;
    }
};
```

454. 4Sum II [Medium]

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500. All integers are in the range of -228 to 228 – 1 and the result is guaranteed to be at most 231 – 1.

Example:

Input: A = [1, 2] B = [-2,-1] C = [-1, 2] D = [0, 2]

Output: 2

Explanation: The two tuples are:

1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0
2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0

题目大意：给四个长度相等的整型数组A、B、C、D，寻找i，j，k，l使得A[i] + B[j] + C[k] + D[l] = 0，求问有多少个这样的i，j，k，l组合～

分析：设立两个map，m1和m2，m1中存储A、B数组中的元素两两组合的和以及出现的次数，如m1[i] = j表示A、B两数组中各取一个元素相加的和为i的个数有j个～这样我们就能把A+B+C+D = 0的问题转化成A+B的和为sum，求C+D中有没有-sum可以满足相加等于0～如果m1[sum]的个数是cnt1，m2[0-sum]的个数是cnt2，那么就能构成cnt1*cnt2个组合～将所有满足条件的结果累加即可得到result的值～

```
class Solution {
public:
    int fourSumCount(vector<int>& A, vector<int>& B, vector<int>& C,
vector<int>& D) {
        map<int, int> m1, m2;
```

```
        int result = 0, n = A.size();
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < n; j++) {
                int t1 = A[i] + B[j];
                int t2 = C[i] + D[j];
                m1[t1]++;
                m2[t2]++;
            }
        }
        for(auto it = m1.begin(); it != m1.end(); it++)
            result += it->second * m2[0 - it->first];
        return result;
    }
};
```

455. Assign Cookies [Easy]

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie. Each child i has a greed factor gi, which is the minimum size of a cookie that the child will be content with; and each cookie j has a size sj. If sj >= gi, we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Note: You may assume the greed factor is always positive. You cannot assign more than one cookie to one child.

Example 1: Input: [1,2,3], [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3. And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content. You need to output 1. Example 2: Input: [1,2], [1,2,3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2. You have 3 cookies and their sizes are big enough to gratify all of the children, You need to output 2.

分析：先将g数组和s数组从小到大排序～ i指针遍历g数组，j指针遍历s数组，如果当前g[i] <= s[j]，也就是当前糖果j能够分给当前小朋友i，那就分配，并且将i指针指向下一个小朋友，cnt同时也要累加一个～如果当前糖果不能分配给当前小朋友，说明糖果j不能分给任何人了，因为没有比小朋友i需要的还要少的小朋友了（i后面的数都比i大）。所以无论是否分配，都把j向后移动一次～这样能保证需求量小的所有小朋友都分配得到能够分配的糖果，此时的cnt也是所求的贪心最大值～避免了大材小用（大糖果分配给需求量小的小朋友）的情况～

```
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
```

```
        sort(g.begin(), g.end());
        sort(s.begin(), s.end());
        int cnt = 0, i = 0, j = 0;
        while(i < g.size() && j < s.size()) {
            if(g[i] <= s[j]) {
                cnt++;
                i++;
            }
            j++;
        }
        return cnt;
    }
};
```

459. Repeated Substring Pattern [Easy]

Given a non-empty string check if it can be constructed by taking a substring of it and appending multiple copies of the substring together. You may assume the given string consists of lowercase English letters only and its length will not exceed 10000.

Example 1: Input: “abab”

Output: True

Explanation: It’s the substring “ab” twice. Example 2: Input: “aba”

Output: False Example 3: Input: “abcabcabcabc”

Output: True

Explanation: It’s the substring “abc” four times. (And the substring “abcabc” twice.)

分析：设字串长度为len，字符串长度为slen，len从1开始一直到slen / 2遍历，如果slen % len != 0肯定当前len不符合，直接continue；否则将每一个长度为len的字串取出到sub2，比较与sub1是否相等，如果所有的都相等说明满足条件，return true，如果到最后循环结束后依旧没有找到这样一个len满足条件，则return false

```
class Solution {
public:
    bool repeatedSubstringPattern(string str) {
        int slen = str.length();
        for(int len = 1; len <= slen / 2; len++) {
            if(slen % len != 0)
                continue;
            string sub1 = str.substr(0, len);
            int flag = 0;
            for(int i = len; i < slen; i += len) {
                string sub2 = str.substr(i, len);
                if(sub1 != sub2) {
                    flag = 1;
```

```
                    break;
                }
            }
            if(flag == 0)
                return true;
        }
        return false;
    }
};
```

461. Hamming Distance [Easy]

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, calculate the Hamming distance.

Note: 0 ≤ x, y < 2^31.

Example:

Input: x = 1, y = 4

Output: 2

Explanation: 1 (0 0 0 1) 4 (0 1 0 0) ↑ ↑

The above arrows point to positions where the corresponding bits are different.

分析：将x和y不断右移一位，然后比较他们的最后一位。对于右移一位，采用x / 2和 y / 2的方式，对于比较最后一位，即比较x % 2 和 y % 2，统计不相同的次数cnt，直到x和y都等于0为止

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int cnt = 0;
        while(x != 0 || y != 0) {
            if(x % 2 != y % 2)
                cnt++;
            x /= 2;
            y /= 2;
        }
        return cnt;
    }
};
```

462. Minimum Moves to Equal Array Elements II [Medium]

Given a non-empty integer array, find the minimum number of moves required to make all array elements equal, where a move is incrementing a selected element by 1 or decrementing a selected element by 1.

You may assume the array's length is at most 10,000.

Example:

Input: [1,2,3]

Output: 2

Explanation: Only two moves are needed (remember each move increments or decrements one element):

[1,2,3] => [2,2,3] => [2,2,2]

题目大意：给一个非空整数数组，每次可以将数组中一个元素+1或者-1，求最少需要多少次这样的操作，才能使数组中所有的数都想等～

分析：让所有数都等于那个第n/2 + 1大的数字～首先用nth_element(nums.begin(), nums.begin() + n / 2, nums.end());将第n/2 + 1大的数字放到最中间，然后取得它的值为mid，最后遍历数组，累加所有元素与mid的差的绝对值即为所求～

```
class Solution {
public:
    int minMoves2(vector<int>& nums) {
        int result = 0, n = nums.size();
        nth_element(nums.begin(), nums.begin() + n / 2, nums.end());
        int mid = *(nums.begin() + n / 2);
        for(int i = 0; i < n; i++)
            result += abs(nums[i] - mid);
        return result;
    }
};
```

463. Island Perimeter [Easy]

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water. Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells). The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example:

[[0,1,0,0], [1,1,1,0], [0,1,0,0], [1,1,0,0]]

Answer: 16

分析：如果彼此不相连，那么一个1就应该有4条边；考虑相连的情况，为了避免重复计算，只取每个为1的坐标的左边和上边： 如果当前点为1而且它上面也为1，他们彼此相连导致双方都失去1条边，也就是2条边；同理如果当前点为1它左边也为1，他们彼此相连导致双方都失去1条边，也就是2条边。不考虑第一行没有上一行和第一列没有左边的情况，则可以遍历每一个格子得到cnt

```
class Solution {
public:
    int islandPerimeter(vector<vector<int>>& grid) {
        int cnt = 0;
        for(int i = 0; i < grid.size(); i++) {
            for(int j = 0; j < grid[i].size(); j++) {
                if(grid[i][j] == 1) {
                    cnt += 4;
                    if(i != 0 && grid[i-1][j] == 1)
                        cnt -= 2;
                    if(j != 0 && grid[i][j-1] == 1)
                        cnt -= 2;
                }
            }
        }
        return cnt;
    }
};
```

464. Can I Win [Medium]

In the “100 game,” two players take turns adding, to a running total, any integer from 1..10. The player who first causes the running total to reach or exceed 100 wins.

What if we change the game so that players cannot re-use integers?

For example, two players might take turns drawing from a common pool of numbers of 1..15 without replacement until they reach a total >= 100.

Given an integer maxChoosableInteger and another integer desiredTotal, determine if the first player to move can force a win, assuming both players play optimally.

You can always assume that maxChoosableInteger will not be larger than 20 and desiredTotal will not be larger than 300.

Example

Input: maxChoosableInteger = 10 desiredTotal = 11

Output: false

Explanation: No matter which integer the first player choose, the first player will lose. The first player can choose an integer from 1 up to 10. If the first player choose 1, the second player can only choose integers from 2 up to 10. The second player will win by choosing 10 and get a total = 11, which is >= desiredTotal. Same with other integers chosen by the first player, the second

player will always win.

题目大意：两个人玩游戏，从1～maxChoosableInteger中任选一个数字，第一个人先选，第二个人后选，每个人选过的数字就不能再选了～两个人谁先加起来总和超过desiredTotal谁就赢，问给出数字，第一个人能否赢得比赛～

分析：两个特殊情况：1.如果能够选的最大数字大于等于desiredTotal，第一个人又不傻肯定选最大的值～那样他就赢啦～即： if(maxn >= desiredTotal) return true; 2.如果所有能够选的数字的总和都小于desiredTotal，再怎么选两个人都不可能赢，所以肯定输～： 总和就是首项加末项的和乘以项数除以2～：if((1 + maxn) * maxn / 2 < desiredTotal) return false; 然后建立一个canWin函数，需要用visited标记某个数字是否被选过～因为可选的数字最大不超过20，则可以用一个整型数组标记，因为整型有32位～如果1被选过就把第1位（不是第0位哦～当然也可以从0开始啦～）标记为1，如果2被选过就把第2位标记为1～这样保证所有数字不重复～ 所以传入两个值，一个是想要达到（或者说超过，也就是大于等于啦）的目标数字target，另一个是visited数字标记哪些数字被选过了～ 用map标记当前情况在map表中是否存在，存在的话结果保存在map里面～如果我们发现这个visited也就是这个数字选择的情况已经被保存过了，就直接返回在map里面保存的结果～ 遍历i从1到maxn（maxn是可以选择的最大值，即maxChoosableInteger），表示考虑选择i的情况，用mask = 1 << i，如果mask和visited进行与运算，如果等于0说明当前的visited没有被访问过，就可以考虑这个i的情况，如果要选的这个i大于target，不傻的这个人就会选择i因为肯定能赢啦～还有种情况自己能赢，就是对方输了，即：canWin(target – i, mask | visited) == false，（mask | visited表示把i那位也标记为1～）这个时候把visited情况存起来并且return true，表示赢了～如果所有数字都遍历完还是没有return true，那就最后return false～return false之前不要忘记把当前状态存储起来～也就是m[visited] = false; ～ 注意：调了两个小时的bug后来才发现！(mask & visited) == 0一开始忘记加括号了，写成了mask & visited == 0，但是&运算符比==优先级低，所以==会先运算。。所以就gg了～注意注意～号外号外～

```
class Solution {
private:
    int maxn;
    map<int, bool> m;
public:
    bool canIWin(int maxChoosableInteger, int desiredTotal) {
        maxn = maxChoosableInteger;
        if(maxn >= desiredTotal) return true;
        if((1 + maxn) * maxn / 2 < desiredTotal) return false;
        return canWin(desiredTotal, 0);
    }
    bool canWin(int target, int visited) {
        if(m.find(visited) != m.end()) return m[visited];
        for(int i = 1; i <= maxn; i++) {
            int mask = (1 << i);
            if((mask & visited) == 0 && (i >= target || canWin(target - i, mask | visited) == false)) {
                m[visited] = true;
                return true;
            }
        }
```

```
            m[visited] = false;
            return false;
        }
};
```

468. Validate IP Address [Medium]

Write a function to check whether an input string is a valid IPv4 address or IPv6 address or neither.

IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots (“.”), e.g.,172.16.254.1;

Besides, leading zeros in the IPv4 is invalid. For example, the address 172.16.254.01 is invalid.

IPv6 addresses are represented as eight groups of four hexadecimal digits, each group representing 16 bits. The groups are separated by colons (“:”). For example, the address 2001:0db8:85a3:0000:0000:8a2e:0370:7334 is a valid one. Also, we could omit some leading zeros among four hexadecimal digits and some low-case characters in the address to upper-case ones, so 2001:db8:85a3:0:0:8A2E:0370:7334 is also a valid IPv6 address(Omit leading zeros and using upper cases).

However, we don’t replace a consecutive group of zero value with a single empty group using two consecutive colons (::) to pursue simplicity. For example, 2001:0db8:85a3::8A2E:0370:7334 is an invalid IPv6 address.

Besides, extra leading zeros in the IPv6 is also invalid. For example, the address 02001:0db8:85a3:0000:0000:8a2e:0370:7334 is invalid.

Note: You may assume there is no extra space or special characters in the input string.

Example 1: Input: “172.16.254.1”

Output: “IPv4”

Explanation: This is a valid IPv4 address, return “IPv4”. Example 2: Input: “2001:0db8:85a3:0:0:8A2E:0370:7334”

Output: “IPv6”

Explanation: This is a valid IPv6 address, return “IPv6”. Example 3: Input: “256.256.256.256”

Output: “Neither”

Explanation: This is neither a IPv4 address nor a IPv6 address.

题目大意：给一个字符串，判断它是不是IP地址，如果是IPv4地址，就返回IPv4；如果是IPv6地址，就返回IPv6；都不是的话返回Neither～

分析：首先根据地址中是否包含.或者:判断是IPv4还是IPv6的范畴～ 如果是IPv4，则需要满足： 1.一共有3个”.” 2.每个点之间的内容为数字，不能为空，且这个数字是0～255 3.没有前导的0，即不会出现001这样的数字（小技巧是字符串转化为数字再转化为字符串，看看和原来字符串是否一致） 如果是IPv6，则需要满足： 1.一共有7个”:” 2.每个标点之间的内容为16进制数字，不能为空即只能包含数字、字母(a~f、A~F)

```cpp
class Solution {
public:
    string validIPAddress(string IP) {
        for (int i = 0; i < IP.length(); i++) {
            if (IP[i] == '.')
                return isIPv4(IP) ? "IPv4" : "Neither";
            else if (IP[i] == ':')
                return isIPv6(IP) ? "IPv6" : "Neither";
        }
        return "Neither";
    }
private:
    bool isIPv4(string IP) {
        int dotcnt = 0;
        for (int i = 0; i < IP.length(); i++) {
            if (IP[i] == '.')
                dotcnt++;
        }
        if (dotcnt != 3) return false;
        string temp = "";
        for (int i = 0; i < IP.length(); i++) {
            if (IP[i] != '.')
                temp += IP[i];
            if (IP[i] == '.' || i == IP.length() - 1) {
                if (temp.length() == 0 || temp.length() > 3) return
false;
                for (int j = 0; j < temp.length(); j++) {
                    if (!isdigit(temp[j])) return false;
                }
                int tempInt = stoi(temp);
                if (tempInt > 255 || tempInt < 0) return false;
                string convertString = to_string(tempInt);
                if (convertString != temp) return false;
                temp = "";
            }
        }
        if (IP[IP.length()-1] == '.') return false;
        return true;
    }

    bool isIPv6(string IP) {
        int dotcnt = 0;
        for (int i = 0; i < IP.length(); i++) {
            if(IP[i] == ':')
                dotcnt++;
        }
        if (dotcnt != 7) return false;
```

```
        string temp = "";
        for (int i = 0; i < IP.length(); i++) {
            if (IP[i] != ':')
                temp += IP[i];
            if (IP[i] == ':' || i == IP.length() - 1) {
                if (temp.length() == 0 || temp.length() > 4) return false;
                for (int j = 0; j < temp.length(); j++) {
                    if (!(isdigit(temp[j]) || (temp[j] >= 'a' && temp[j] <= 'f') || (temp[j] >= 'A' && temp[j] <= 'F'))) return false;
                }
                temp = "";
            }
        }
        if (IP[IP.length()-1] == ':') return false;
        return true;
    }
};
```

475. Heaters [Easy]

Winter is coming! Your first job during the contest is to design a standard heater with fixed warm radius to warm all the houses.

Now, you are given positions of houses and heaters on a horizontal line, find out minimum radius of heaters so that all houses could be covered by those heaters.

So, your input will be the positions of houses and heaters seperately, and your expected output will be the minimum radius standard of heaters.

Note: Numbers of houses and heaters you are given are non-negative and will not exceed 25000. Positions of houses and heaters you are given are non-negative and will not exceed 10^9. As long as a house is in the heaters' warm radius range, it can be warmed. All the heaters follow your radius standard and the warm radius will the same. Example 1: Input: [1,2,3],[2] Output: 1 Explanation: The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed. Example 2: Input: [1,2,3,4],[1,4] Output: 1 Explanation: The two heater was placed in the position 1 and 4. We need to use radius 1 standard, then all the houses can be warmed.

题目大意：给出房子的坐标和供暖器的坐标，求供暖器的最小供热半径是多少才能满足让所有房子都暖和。

分析：先将houses和heaters排序，计算每一个house左右的供暖器的距离最小的那个值，然后将所有的这些最小值中取最大的值。 因为houses和heaters都是排序好的，所以heater[j]与houses[i]的距离应该越来越小，如果突然间变大了，说明不是最小值了，就break掉。这样就能得知最小值。

```
class Solution {
public:
    int findRadius(vector<int>& houses, vector<int>& heaters) {
```

```
        sort(houses.begin(), houses.end());
        sort(heaters.begin(), heaters.end());
        int startindex = 0, maxn = 0;
        for(int i = 0; i < houses.size(); i++) {
            int tempmin = INT_MAX;
            for(int j = startindex; j < heaters.size(); j++) {
                if(abs(heaters[j] - houses[i]) <= tempmin) {
                    tempmin = abs(heaters[j] - houses[i]);
                    startindex = j;
                } else {
                    break;
                }
            }
            maxn = max(maxn, tempmin);
        }
        return maxn;
    }
};
```

476. Number Complement [Easy]

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Note: The given integer is guaranteed to fit within the range of a 32-bit signed integer. You could assume no leading zero bit in the integer’s binary representation. Example 1: Input: 5 Output: 2 Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2. Example 2: Input: 1 Output: 0 Explanation: The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

分析：mask – 1为和num二进制位等长的所有位数为1的数，与num取^可以得到和num相反的数字。

```
class Solution {
public:
    int findComplement(int num) {
        int temp = num, mask = 1;
        while(temp != 0) {
            temp = temp >> 1;
            mask = mask << 1;
        }
        return num ^ (mask - 1);
    }
};
```

477. Total Hamming Distance [Medium]

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

Example: Input: 4, 14, 2

Output: 6

Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just showing the four bits relevant in this case). So the answer will be: HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6. Note: Elements of the given array are in the range of 0 to 10^9 Length of the array will not exceed 10^4.

题目大意：两个数字的二进制表示中，相应的位不相同的个数即为这两个数的Hamming距离。给出一组数，计算他们每两对构成的Hamming距离的总和～

分析：遍历数组中所有数的相同某位，如果有cnt个1，n-cnt个0，则这一位能够构成的Hamming距离为cnt * (n – cnt)～所以从第0位开始一直计算到第32位，将所有位贡献的Hamming距离总和相加即可得到总的Hamming距离～

```
class Solution {
public:
    int totalHammingDistance(vector<int>& nums) {
        int result = 0, n = nums.size();
        for(int i = 0; i < 32; i++) {
            int cnt = 0;
            for(int j = 0; j < n; j++) {
                if(nums[j] >> i & 1 == 1)
                    cnt++;
            }
            result += cnt * (n - cnt);
        }
        return result;
    }
};
```

479. Largest Palindrome Product [Easy]

Find the largest palindrome made from the product of two n-digit numbers. Since the result could be very large, you should return the largest palindrome mod 1337. Example: Input: 2 Output: 987 Explanation: 99 x 91 = 9009, 9009 % 1337 = 987 Note: The range of n is [1,8].

题目大意：找到由两个n位数字乘积得到的最大回文。由于结果可能非常大，返回最大回文的%1337后的结果。

分析：l是两位数的最小值，r是两位数的最大值，从r到1能够组成的回文是r的字符串+r倒置后的字符串组成的字符串，将这个回文字串转为long型的ans，j从r到根号ans中，判断是否有j可以满足能被ans整除的j，并且这个除以的结果是小于r的，如果找到了返回ans%1337的结果，因为如果n == 1的时候比较特殊，组成的回文串是9，所以如果for循环后依然没有返回，说明遇到了n=1，那么最后就

单独返回9～

```cpp
class Solution {
public:
    int largestPalindrome(int n) {
        int l = pow(10, n-1), r = pow(10, n) - 1;
        for (int i = r; i >= 1; i--) {
            string s = to_string(i);
            string t = s;
            reverse(t.begin(), t.end());
            long ans = stol(s + t);
            for (long j = r; j * j >= ans; j--)
                if (ans % j == 0 && ans / j <= r) return ans % 1337;
        }
        return 9;
    }
};
```

481. Magical String [Medium]

A magical string S consists of only '1' and '2' and obeys the following rules:

The string S is magical because concatenating the number of contiguous occurrences of characters '1' and '2' generates the string S itself.

The first few elements of string S is the following: S = "1221121221221121122....."

If we group the consecutive '1's and '2's in S, it will be:

1 22 11 2 1 22 1 22 11 2 11 22 .....

and the occurrences of '1's or '2's in each group are:

1 2 2 1 1 2 1 2 2 1 2 2 .....

You can see that the occurrence sequence above is the S itself.

Given an integer N as input, return the number of '1's in the first N number in the magical string S.

Note: N will not exceed 100,000.

Example 1: Input: 6 Output: 3 Explanation: The first 6 elements of magical string S is "12211" and it contains three 1's, so return 3.

分析：直接按照这个字符串的构造方法还原这个字符串s：首先初始化s = "122",让index指向下标为2处，开始根据index指向的字符在s后面添加字符串，如果指向的是2就添加2个，如果指向的是1就添加一个，具体添加什么字符以当前s的末尾一位的字符是1还是2为准，如果s当前最后一个字符是1就添加2，反之添加1～还原好了之后用count直接计算字符串从begin()到n长度处一共有多少个'1'字符～～

```cpp
class Solution {
public:
    int magicalString(int n) {
        string s = "122";
        int index = 2;
        while(s.length() < n) {
            int cnt = s[index] - '0';
            char c = (s.back() == '1' ? '2' : '1');
            string temp(cnt, c);
            s += temp;
            index++;
        }
        return count(s.begin(), s.begin() + n, '1');
    }
};
```

482. License Key Formatting [Medium]

Now you are given a string S, which represents a software license key which we would like to format. The string S is composed of alphanumerical characters and dashes. The dashes split the alphanumerical characters within the string into groups. (i.e. if there are M dashes, the string is split into M+1 groups). The dashes in the given string are possibly misplaced.

We want each group of characters to be of length K (except for possibly the first group, which could be shorter, but still must contain at least one character). To satisfy this requirement, we will reinsert dashes. Additionally, all the lower case letters in the string must be converted to upper case.

So, you are given a non-empty string S, representing a license key to format, and an integer K. And you need to return the license key formatted according to the description above.

Example 1: Input: S = “2-4A0r7-4k”, K = 4

Output: “24A0-R74K”

Explanation: The string S has been split into two parts, each part has 4 characters. Example 2: Input: S = “2-4A0r7-4k”, K = 3

Output: “24-A0R-74K”

Explanation: The string S has been split into three parts, each part has 3 characters except the first part as it could be shorter as said above. Note: The length of string S will not exceed 12,000, and K is a positive integer. String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9) and dashes(-). String S is non-empty.

题目大意：将序列号重置格式，首先去掉原先有的“-”，然后所有字母必须大写，而且要重新将每K位加一个“-”，如果序列号不能被K整除，则字符串开头允许小于K位～

分析： 1.将“-”去除，且用toupper将所有字符变成大些形式，存储在新的字符串temp里面； 2.先处理首部，如果len % K不等于0，就先将前面len%K个字母放入result字符串中； 3.每K位增加一个“-”，注意：如果一开始index == 0，即len能够被K整除，前面没有多余的，则不需要增加那个多余的第一个“-”～～

```cpp
class Solution {
public:
    string licenseKeyFormatting(string S, int K) {
        string result = "", temp = "";
        for(int i = 0; i < S.length(); i++) {
            if(S[i] != '-')
                temp += toupper(S[i]);
        }
        int len = temp.length(), index = 0;
        while(index < len % K)
            result += temp[index++];
        for(int i = 0; i < len - len % K; i++) {
            if(i % K == 0 && index != 0)
                result += '-';
            result += temp[index++];
        }
        return result;
    }
};
```

485. Max Consecutive Ones [Easy]

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1: Input: [1,1,0,1,1,1] Output: 3 Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3. Note:

The input array will only contain 0 and 1. The length of input array is a positive integer and will not exceed 10,000

分析：设立cnt数组，表示在nums[i]处当前连续的1的值，maxn取其最大的值，在遇到nums[i] == 0的时候更新maxn的值。最后还要更新一下防止最后一个是1.

```cpp
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        vector<int> cnt(nums.size());
        int maxn = 0;
        cnt[0] = nums[0];
        for(int i = 1; i < nums.size(); i++) {
            if(nums[i] == 0) {
                cnt[i] = 0;
                maxn = max(maxn, cnt[i-1]);
            } else {
                cnt[i] = cnt[i-1] + 1;
            }
```

```
        }
        maxn = max(maxn, cnt[nums.size() - 1]);
        return maxn;
    }
};
```

486. Predict the Winner [Medium]

Given an array of scores that are non-negative integers. Player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on. Each time a player picks a number, that number will not be available for the next player. This continues until all the scores have been chosen. The player with the maximum score wins.

Given an array of scores, predict whether player 1 is the winner. You can assume each player plays to maximize his score.

Example 1: Input: [1, 5, 2] Output: False Explanation: Initially, player 1 can choose between 1 and 2. If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2). So, final score of player 1 is 1 + 2 = 3, and player 2 is 5. Hence, player 1 will never be the winner and you need to return False. Example 2: Input: [1, 5, 233, 7] Output: True Explanation: Player 1 first chooses 1. Then player 2 have to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233. Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win. Note: 1 <= length of the array <= 20. Any scores in the given array are non-negative integers and will not exceed 10,000,000. If the scores of both players are equal, then player 1 is still the winner.

题目大意：给一个整型数组，两个人依次从数组中的头或者尾拿一个数，判断是否player1拿到的总数大于或者等于player2～如果是就返回true，否则返回false～

分析：动态规划解决～建立dp[len][len]数组，dp[i][j]表示nums数组中i～j下标间player1能够获得的分数-player2能够获得的分数～最后dp[0][len-1]的正负性即可表明player1是否能赢～ dp[0][len-1]的值通过递归动态规划可得：func(begin, end)返回dp[begin][end]的值，当begin和end相等的时候，dp[begin][end]的值即为nums[begin]（或者nums[end]），如果begin和end不等，那么如果取begin，结果为nums[begin] – dp[begin+1][end]; 如果取end，结果为nums[end] – dp[begin][end-1]，dp[begin][end]取它俩中较大的一个～

```
class Solution {
public:
    bool PredictTheWinner(vector<int>& nums) {
        this->nums = nums;
        int len = nums.size();
        dp.resize(len, vector<int>(len));
        return func(0, len-1) >= 0;
    }
private:
    vector<int> nums;
    vector<vector<int>> dp;
```

```
    int func(int begin, int end) {
        dp[begin][end] = begin == end ? nums[begin] : max(nums[begin] - func(begin+1, end), nums[end] - func(begin, end-1));
        return dp[begin][end];
    }
};
```

491. Increasing Subsequences [Medium]

Given an integer array, your task is to find all the different possible increasing subsequences of the given array, and the length of an increasing subsequence should be at least 2 .

Example: Input: [4, 6, 7, 7] Output: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]] Note: The length of the given array will not exceed 15. The range of integer in the given array is [-100,100]. The given array may contain duplicates, and two equal integers should also be considered as a special case of increasing sequence.

题目大意：给一个整型数组nums，寻找它的所有满足条件的子数组：每一个子数组内的元素都是递增的（允许包含重复元素，所以其实是不递减的）～以二维数组的方式返回～

分析：row为结果集的每一行的结果，深度优先搜索，两个参数：lastNum：表示当前row中最后一个元素的值，（即后面想要加进来的值必须大于等于lastNum），index：表示当前已加入row的元素中的最后一位的index，一开始lastNum = -101表示最小值，而index = -1。这样下一次就让i从index + 1开始遍历是否有比它大的或者等于它的元素，有就加入row并且dfs(nums[i], i)，最后记得将row的最后一个元素pop出来。当row中的元素大于等于2个的时候，就可以称为一个满足条件的结果，因为为了避免有重复元素，就将row插入一个集合中。最后遍历集合将所有一维数组都放入result二维数组中，返回result即可～

```
class Solution {
public:
    vector<vector<int>> findSubsequences(vector<int>& nums) {
        vector<vector<int>> result;
        this->nums = nums;
        dfs(-101, -1);
        for (auto it = s.begin(); it != s.end(); it++)
            result.push_back(*it);
        return result;
    }
private:
    set<vector<int>> s;
    vector<int> nums, row;
    void dfs(int lastNum, int index) {
        if (row.size() >= 2) s.insert(row);
        if (index == nums.size()) return ;
        for (int i = index + 1; i < nums.size(); i++) {
            if (nums[i] >= lastNum) {
                row.push_back(nums[i]);
                dfs(nums[i], i);
```

```
                row.pop_back();
            }
        }
    }
};
```

492. Construct the Rectangle [Medium]

For a web developer, it is very important to know how to design a web page's size. So, given a specific rectangular web page's area, your job by now is to design a rectangular web page, whose length L and width W satisfy the following requirements:

1. The area of the rectangular web page you designed must equal to the given target area.
2. The width W should not be larger than the length L, which means L >= W.
3. The difference between length L and width W should be as small as possible. You need to output the length L and the width W of the web page you designed in sequence. Example: Input: 4 Output: [2, 2] Explanation: The target area is 4, and all the possible ways to construct it are [1,4], [2,2], [4,1]. But according to requirement 2, [1,4] is illegal; according to requirement 3, [4,1] is not optimal compared to [2,2]. So the length L is 2, and the width W is 2. Note: The given area won't exceed 10,000,000 and is a positive integer The web page's width and length you designed must be positive integers.

题目大意：给定area，求长和宽，使得长方形面积等于area，要求长>=宽并且长和宽之间的大小差距尽可能小～～

分析：先令长l和宽w等于sqrt(area), 如果长x宽得到的面积不等于area，稍微调整l和w的大小：如果面积小了，将长+1；如果面积大了，将宽-1。直到最后l * w == area为止～

```
class Solution {
public:
    vector<int> constructRectangle(int area) {
        vector<int> result(2, 0);
        int l = sqrt(area), w = sqrt(area);
        while(l * w != area) {
            if(l * w < area)
                l++;
            else
                w--;
        }
        result[0] = l;
        result[1] = w;
        return result;
    }
};
```

494. Target Sum [Medium]

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols + and -. For each integer, you should choose one from + and – as its new symbol.

Find out how many ways to assign symbols to make sum of integers equal to target S.

Example 1: Input: nums is [1, 1, 1, 1, 1], S is 3. Output: 5 Explanation:

-1+1+1+1+1 = 3 +1-1+1+1+1 = 3 +1+1-1+1+1 = 3 +1+1+1-1+1 = 3 +1+1+1+1-1 = 3

There are 5 ways to assign symbols to make the sum of nums be target 3. Note: The length of the given array is positive and will not exceed 20. The sum of elements in the given array will not exceed 1000. Your output answer is guaranteed to be fitted in a 32-bit integer.

题目大意：给一个非负的整数数组，可以在每个元素前面加+或者-，给一个目标整数S，求问有多少种不同的添加方式，可以让数组中所有元素在添加符号之后相加的结果为S～

分析：深度优先搜索，尝试每次添加+或者-，当当前cnt为nums数组的大小的时候，判断sum与S是否相等，如果相等就result++。sum为当前cnt步数情况下的前面所有部分的总和～

```
class Solution {
public:
    int result;
    int findTargetSumWays(vector<int>& nums, int S) {
        dfs(0, 0, nums, S);
        return result;
    }
    void dfs(int sum, int cnt, vector<int>& nums, int S) {
        if(cnt == nums.size()) {
            if(sum == S)
                result++;
            return ;
        }
        dfs(sum + nums[cnt], cnt + 1, nums, S);
        dfs(sum - nums[cnt], cnt + 1, nums, S);
    }
};
```

495. Teemo Attacking [Medium]

In LLP world, there is a hero called Teemo and his attacking can make his enemy Ashe be in poisoned condition. Now, given the Teemo's attacking ascending time series towards Ashe and the poisoning time duration per Teemo's attacking, you need to output the total time that Ashe is in poisoned condition.

You may assume that Teemo attacks at the very beginning of a specific time point, and makes Ashe be in poisoned condition immediately.

Example 1: Input: [1,4], 2 Output: 4 Explanation: At time point 1, Teemo starts attacking Ashe and makes Ashe be poisoned immediately. This poisoned status will last 2 seconds until the end of time point 2. And at time point 4, Teemo attacks Ashe again, and causes Ashe to be in poisoned status for another 2 seconds. So you finally need to output 4. Example 2: Input: [1,2], 2 Output: 3 Explanation: At time point 1, Teemo starts attacking Ashe and makes Ashe be poisoned. This poisoned status will last 2 seconds until the end of time point 2. However, at the beginning of time point 2, Teemo attacks Ashe again who is already in poisoned status. Since the poisoned status won't add up together, though the second poisoning attack will still work at time point 2, it will stop at the end of time point 3. So you finally need to output 3. Note: You may assume the length of given time series array won't exceed 10000. You may assume the numbers in the Teemo's attacking time series and his poisoning time duration per attacking are non-negative integers, which won't exceed 10,000,000.

题目大意：给一个数组和一个数字，数组中元素是升序的，每一个元素代表释放毒气的时间点，数字duration表示释放毒气能够中毒的时间，比如时间点1开始攻击，持续2秒，则在第2秒结束后才会恢复不中毒～如果前一次中毒ing而后一次提早又被攻击，那就是后一次攻击后+持续时间才会恢复不中毒～现在要求这个人一共中毒了多少秒～

分析：oldEnd表示这一次攻击之前的攻击导致的最晚中毒结束时间，首先初值是-1表示刚开始的时候不是中毒的（也就是中毒结束时间是-1，不能是0因为有一个测试用例时间点是从0开始的。。） 遍历数组，对于每一个timeSeries[i], 这次攻击导致的中毒结束时间是newEnd = timeSeries[i] + duration – 1： 如果新的结束时间比之前的结束时间长，就取duration, newEnd – oldEnd)中较小的一个，因为如果oldEnd早就结束了，那就是duration让他中毒了duration秒，如果oldEnd结束的比较晚，晚于这一次攻击开始的时间，那么累计的中毒事件就应该是newEnd – oldEnd也就是新的攻击造成的结束时间减去旧的攻击结束时间～并且让oldEnd更新为newEnd的这个较大的值～ 如果新的结束时间比之前的结束时间短，那么不需要累加，反正这次攻击不攻击都没什么用因为反正是中毒ing...最后返回result累加的结果即可～

```
class Solution {
public:
    int findPoisonedDuration(vector<int>& timeSeries, int duration) {
        int result = 0, oldEnd = -1;
        for (int i = 0; i < timeSeries.size(); i++) {
            int newEnd = timeSeries[i] + duration - 1;
            if (newEnd > oldEnd) {
                result += min(duration, newEnd - oldEnd);
                oldEnd = newEnd;
            }
        }
        return result;
    }
};
```

496. Next Greater Element I [Easy]

You are given two arrays (without duplicates) nums1 and nums2 where nums1's elements are subset of nums2. Find all the next greater numbers for nums1's elements in the corresponding places of nums2.

The Next Greater Number of a number x in nums1 is the first greater number to its right in nums2. If it does not exist, output -1 for this number.

Example 1: Input: nums1 = [4,1,2], nums2 = [1,3,4,2]. Output: [-1,3,-1] Explanation: For number 4 in the first array, you cannot find the next greater number for it in the second array, so output -1. For number 1 in the first array, the next greater number for it in the second array is 3. For number 2 in the first array, there is no next greater number for it in the second array, so output -1. Example 2: Input: nums1 = [2,4], nums2 = [1,2,3,4]. Output: [3,-1] Explanation: For number 2 in the first array, the next greater number for it in the second array is 3. For number 4 in the first array, there is no next greater number for it in the second array, so output -1. Note: All elements in nums1 and nums2 are unique. The length of both nums1 and nums2 would not exceed 1000.

题目大意：给两个数组findNums和nums，返回一个和findNums等长的数组，其返回的内容为：对于findNums[i]，返回的result[i]为nums[i]数组中，findNums[i]的后面的元素中第一个比findNums[i]大的元素。如果不存在这样的元素就写-1。其中findNums数组是nums数组的子数组～

分析：使用栈，从后往前遍历nums[i]，每当栈不为空的时候，一直出栈直到遇到比nums[i]大的数字停止。设立一个map<int, int> m，存储nums中每一个元素以及它对应的下一个最大元素构成的映射。如果停止后栈为空就将m[nums[i]]标记为-1，否则就写栈的栈顶元素～ 最后将findNums中出现的每一个元素对应的map的值放入result数组中返回～

```
class Solution {
public:
    vector<int> nextGreaterElement(vector<int>& findNums, vector<int>&
nums) {
        vector<int> result;
        stack<int> s;
        map<int, int> m;
        for (int i = nums.size() - 1; i >= 0; i--) {
            while (!s.empty() && s.top() <= nums[i])
                s.pop();
            m[nums[i]] = s.empty() ? -1 : s.top();
            s.push(nums[i]);
        }
        for (int i = 0; i < findNums.size(); i++)
            result.push_back(m[findNums[i]]);
        return result;
    }
};
```

498. Diagonal Traverse [Medium]

Given a matrix of M x N elements (M rows, N columns), return all elements of the matrix in diagonal order as shown in the below image.

Example: Input: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] Output: [1,2,4,7,5,3,6,8,9] Explanation:

Note: The total number of elements of the given matrix will not exceed 10,000.

题目大意：按照如图所示方向遍历一个m*n的二维数组，将元素按序放入一维数组中返回～

分析：先假设所有的方向都是斜向上的，也就是从matrix[0][0]开始，matrix[1][0]、matrix[2, 0]... matrix[m-1][0]也就是第一列的所有数为开头，然后还有最后一行的所有数为开头，寻找matrix[i-1][j+1]是否存在，存在就将它放入temp数组中，temp的每一行表示斜着的一行元素的序列。 这样当temp数组的行数是奇数的时候，就从后往前一次放入到result数组中，如果是偶数就从前往后依次放入result数组中，返回result数组即为所求～

```
class Solution {
public:
    vector<int> findDiagonalOrder(vector<vector<int>>& matrix) {
        vector<int> result;
        int m = matrix.size();
        if (m == 0) return result;
        int n = matrix[0].size(), index = 0;
        vector<vector<int>> temp(m + n - 1);
        for (int i = 0; i < m; i++, index++) {
            temp[index].push_back(matrix[i][0]);
            for (int x = i, y = 0; x - 1 >= 0 && y + 1 < n; x--, y++)
                temp[index].push_back(matrix[x-1][y+1]);
        }
        for (int j = 1; j < n; j++, index++) {
            temp[index].push_back(matrix[m-1][j]);
            for (int x = m - 1, y = j; x - 1 >= 0 && y + 1 < n; x--,
y++)
                temp[index].push_back(matrix[x-1][y+1]);
        }
        for (int i = 0; i < m + n - 1; i++) {
            if (i % 2 == 1) {
                for (int j = temp[i].size() - 1; j >= 0; j--)
                    result.push_back(temp[i][j]);
            } else {
                for (int j = 0; j < temp[i].size(); j++)
                result.push_back(temp[i][j]);
            }
        }
        return result;
    }
};
```

500. Keyboard Row [Easy]

Given a List of words, return the words that can be typed using letters of alphabet on only one row's of American keyboard like the image below. Example 1: Input: ["Hello", "Alaska", "Dad", "Peace"] Output: ["Alaska", "Dad"] Note: You may use one character in the keyboard more than once. You may assume the input string will only contain letters of alphabet.

题目大意：给一个单词数组，判断哪些单词是可以由键盘的一行中的字母构成的，返回这些单词～

分析：设立一个集合数组，v[0]、v[1]、v[2]集合分别插入键盘的第1～3行的所有字母的集合（大小写都包括），接着遍历每一个单词，首先判断单词的第一个字母是处于哪一行的，tag表示其所属行数的下标，接着对于单词的每一个字母，判断是否在v[tag]这个集合里面，如果所有的都存在就将这个单词放入result数组中返回～

```
class Solution {
public:
    vector<string> findWords(vector<string>& words) {
        vector<string> result;
        vector<set<char>> v(3);
        string s1 = "QWERTYUIOPqwertyuiop", s2 = "ASDFGHJKLasdfghjkl",
s3 = "ZXCVBNMzxcvbnm";
        for (int i = 0; i < s1.length(); i++) v[0].insert(s1[i]);
        for (int i = 0; i < s2.length(); i++) v[1].insert(s2[i]);
        for (int i = 0; i < s3.length(); i++) v[2].insert(s3[i]);
        for (int i = 0; i < words.size(); i++) {
            int tag = -1;
            bool flag = true;
            if (words[i].length() == 0) continue;
            if (v[0].find(words[i][0]) != v[0].end()) tag = 0;
            if (v[1].find(words[i][0]) != v[1].end()) tag = 1;
            if (v[2].find(words[i][0]) != v[2].end()) tag = 2;
            for (int j = 1; j < words[i].length(); j++) {
                if (v[tag].find(words[i][j]) == v[tag].end()) {
                    flag = false;
                    break;
                }
            }
            if (flag == true)
                result.push_back(words[i]);
        }
        return result;
    }
};
```

501. Find Mode in Binary Search Tree [Medium]

Given a binary search tree (BST) with duplicates, find all the mode(s) (the most frequently occurred element) in the given BST.

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than or equal to the node's key. The right subtree of a node contains only nodes with keys greater than or equal to the node's key. Both the left and right subtrees must also be binary search trees. For example: Given BST [1,null,2,2], 1 \ 2 / 2 return [2].

Note: If a tree has more than one mode, you can return them in any order.

Follow up: Could you do that without using any extra space? (Assume that the implicit stack space incurred due to recursion does not count).

题目大意：给一个二叉搜索树，求出现次数最多的数是哪个（哪些）～

分析：二叉搜索树的中序遍历的结果恰好是所有数的递增序列，根据中序遍历结果，对于当前遍历结点，标记maxCount为最大出现次数，tempCount为当前数字出现的次数，currentVal为当前保存的值。 首先，tempCount++表示当前的数字出现次数+1，如果当前结点的值不等于保存的值，就更新currentVal的值，并且将tempCount标记为1～ 接下来，如果tempCount即当前数字出现的次数大于maxCount，就更新maxCount，并且将result数组清零，并将当前数字放入result数组中；如果tempCount只是等于maxCount，说明是出现次数一样的，则将当前数字直接放入result数组中～

```
class Solution {
public:
    vector<int> findMode(TreeNode* root) {
        inorder(root);
        return result;
    }
private:
    vector<int> result;
    int maxCount = 0, currentVal, tempCount = 0;
    void inorder(TreeNode* root) {
        if (root == NULL) return;
        inorder(root->left);
        tempCount++;
        if (root->val != currentVal) {
            currentVal = root->val;
            tempCount = 1;
        }
        if (tempCount > maxCount) {
            maxCount = tempCount;
            result.clear();
            result.push_back(root->val);
        } else if (tempCount == maxCount) {
            result.push_back(root->val);
        }
        inorder(root->right);
    }
};
```

503. Next Greater Element II [Medium]

Given a circular array (the next element of the last element is the first element of the array), print the Next Greater Number for every element. The Next Greater Number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, output -1 for this number.

Example 1: Input: [1,2,1] Output: [2,-1,2] Explanation: The first 1's next greater number is 2; The number 2 can't find next greater number; The second 1's next greater number needs to search circularly, which is also 2.

题目大意：给一个循环数组，返回一个等长的数组，数组中的每一个元素是：它后面的第一个大于它的元素（如果后面没有就循环一遍到最前面找，直到循环了一圈为止），如果不存在这样的数，就返回-1～

分析：首先建立一个等长的result数组，起始都是-1。和Next Greater Element I类似，不过这个题目要两个循环解决，第一个循环i从0～n-1，对于每一个nums[i]，把他们的下标index都放入栈中。但是在放入栈之前需要做这样的事情：比较栈顶元素和nums[i]，如果恰好比nums[i]小，说明nums[i]就是它的第一大元素，就将result[s.top()]的值变为nums[i]，这样栈中的下标始终是它的后面找不到比他大的元素的下标，也就是说栈中在遍历一遍后剩下的元素的值都是递减次序的～ 开始第二次循环，依旧i从0～n-1，但是这次不需要push入栈了，只需要处理栈中剩下的元素，对于nums[i]，如果栈顶元素和nums[i]比较，恰好nums[i]大，说明nums[i]就是他们这些没在后面找到最大元素的最大元素，出栈，result[s.top()] = nums[i]。这样所有遍历完毕后栈中只会剩下唯一一个元素，也就是该数组中最大的元素，它的result依旧是-1，其他的都已经更新完毕。也就是说，第一次遍历是为了找它后面比它大的元素，而第二次遍历是为了找前面比他大的元素～

```
class Solution {
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        int n = nums.size();
        vector<int> result(n, -1);
        stack<int> s;
        for (int i = 0; i < n; i++) {
            while (!s.empty() && nums[s.top()] < nums[i]) {
                result[s.top()] = nums[i];
                s.pop();
            }
            s.push(i);
        }
        for (int i = 0; i < n; i++) {
            while (!s.empty() && nums[s.top()] < nums[i]) {
                result[s.top()] = nums[i];
                s.pop();
            }
        }
        return result;
    }
};
```

504. Base 7 [Easy]

Given an integer, return its base 7 string representation.

Example 1: Input: 100 Output: “202” Example 2: Input: -7 Output: “-10” Note: The input will be in range of [-1e7, 1e7].

题目大意：给一个整数，以字符串形式返回它的7进制～

分析：如果num等于0则直接return 0，如果num小于0则变为相反数，且标记sign为-，每次将对num 取余 7的结果插入result字符串的最前面，并将num / 7，最后返回sign + result的字符串结果～

```
class Solution {
public:
    string convertToBase7(int num) {
        if (num == 0) return "0";
        string sign = "";
        if (num < 0) {
            num = 0 - num;
            sign = "-";
        }
        string result = "";
        while (num != 0) {
            result = to_string(num % 7) + result;
            num = num / 7;
        }
        return sign + result;
    }
};
```

506. Relative Ranks [Easy]

Given scores of N athletes, find their relative ranks and the people with the top three highest scores, who will be awarded medals: “Gold Medal”, “Silver Medal” and “Bronze Medal”.

Example 1: Input: [5, 4, 3, 2, 1] Output: [“Gold Medal”, “Silver Medal”, “Bronze Medal”, “4”, “5”] Explanation: The first three athletes got the top three highest scores, so they got “Gold Medal”, “Silver Medal” and “Bronze Medal”. For the left two athletes, you just need to output their relative ranks according to their scores. Note: N is a positive integer and won’t exceed 10,000. All the scores of athletes are guaranteed to be unique.

题目大意：给一个数组，为几个运动员的分数，返回他们的排名，如果前三名应该为”Gold Medal”, “Silver Medal”, “Bronze Medal”，否则是数字名次，保证所有的分数不重复～

分析：拷贝一个相同的数组arr，然后排序，从nums[i]中寻找与arr[j]相同的分数，名次即为j+1。如果j为0，1，2则需要特殊处理～

```cpp
class Solution {
public:
    vector<string> findRelativeRanks(vector<int>& nums) {
        vector<int> arr = nums;
        auto cmp = [](int a, int b) {return a > b;};
        sort(arr.begin(), arr.end(), cmp);
        vector<string> result(nums.size());
        for (int i = 0; i < nums.size(); i++) {
            for (int j = 0; j < nums.size(); j++) {
                if (nums[i] == arr[j]) {
                    switch(j) {
                        case 0: result[i] = "Gold Medal"; break;
                        case 1: result[i] = "Silver Medal"; break;
                        case 2: result[i] = "Bronze Medal"; break;
                        default: result[i] = to_string(j + 1); break;
                    }
                    break;
                }
            }
        }
        return result;
    }
};
```

507. Perfect Number [Easy]

We define the Perfect Number is a positive integer that is equal to the sum of all its positive divisors except itself.

Now, given an integer n, write a function that returns true when it is a perfect number and false when it is not. Example: Input: 28 Output: True Explanation: 28 = 1 + 2 + 4 + 7 + 14 Note: The input number n will not exceed 100,000,000. (1e8)

题目大意：完美数字是指它的所有可以整除的正数中除了它本身，其他数字之和等于这个数字的数。给一个正整数n，写一个函数，当它是一个完美数字的时候返回true否则false。

分析：从2～sqrt(num)，累加所有i和num/i【因为如果从1～num一个个试是否可以整除的话会超时，而且也没必要，因为知道了除数a必然就知道了num/a这个数字也是它的除数】因为最后还有一个1没有加，所以sum一开始为1，然后返回num == sum，注意如果num本身为1，则要return false，因为1的唯一一个除数1是它本身不能累加，所以1不满足条件～

```cpp
class Solution {
public:
    bool checkPerfectNumber(int num) {
        if (num == 1) return false;
        int sum = 1;
        for (int i = 2; i <= sqrt(num); i++)
            if (num % i == 0) sum = sum + (num / i) + i;
        return num == sum;
    }
};
```

508. Most Frequent Subtree Sum [Medium]

Given the root of a tree, you are asked to find the most frequent subtree sum. The subtree sum of a node is defined as the sum of all the node values formed by the subtree rooted at that node (including the node itself). So what is the most frequent subtree sum value? If there is a tie, return all the values with the highest frequency in any order.

Examples 1 Input:

5 / \ 2 -3 return [2, -3, 4], since all the values happen only once, return all of them in any order. Examples 2 Input:

5 / \ 2 -5 return [2], since 2 happens twice, however -5 only occur once. Note: You may assume the sum of values in any subtree is in the range of 32-bit signed integer.

题目大意：给一棵树，找出现次数最多的子树和。子树和就是一个结点以及它下方所有结点构成的子树的总和，在这些总和中找一个出现次数最多的结果，如果有很多个这样的结果，返回这些结果构成的数组～

分析：首先深度优先搜索，先遍历左子树再遍历右子树，遍历的同时更新子树的根结点的值为自身+左子树和+右子树和。最后将这个子树根结点的值放入一个map中++，表示这个值出现的次数+1。这样深度优先搜索后的树的每一个结点的值都是子树和的值～ 然后遍历这个map，找到最大值maxn。遍历map将出现次数为maxn的所有值放入result数组中，返回result数组即为所求～

```cpp
class Solution {
private:
    map<int, int> m;
public:
    vector<int> findFrequentTreeSum(TreeNode* root) {
        vector<int> result;
        dfs(root);
        int maxn = 0;
        for (auto it = m.begin(); it != m.end(); it++)
            maxn = max(maxn, it->second);
        for (auto it = m.begin(); it != m.end(); it++)
            if (it->second == maxn)
                result.push_back(it->first);
```

```
        return result;
    }

    void dfs(TreeNode* root) {
        if (root == NULL) return ;
        if (root->left != NULL) {
            dfs(root->left);
            root->val += root->left->val;
        }
        if (root->right != NULL) {
            dfs(root->right);
            root->val += root->right->val;
        }
        m[root->val]++;
    }
};
```

513. Find Bottom Left Tree Value [Medium]

Given a binary tree, find the leftmost value in the last row of the tree. Example 1: Input: 2 / \ 1 3 Output: 1 Example 2: Input: 1 / \ 2 3 / / \ 4 5 6 / 7 Output: 7

题目大意：给一个二叉树，找它的最后一行的最左边的结点的值～

分析：广度优先搜索，对每一层进行遍历，result每次保存每一层的第一个值，最后层序遍历完成之后的result即为最后一行的第一个结点的值～

```
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        int result = root->val;
        queue<TreeNode *> q;
        q.push(root);
        TreeNode* temp;
        while (!q.empty()) {
            int size = q.size();
            result = q.front()->val;
            while (size--) {
                temp = q.front();
                q.pop();
                if (temp->left != NULL) q.push(temp->left);
                if (temp->right != NULL) q.push(temp->right);
            }
        }
        return result;
    }
};
```

515. Find Largest Value in Each Tree Row [Medium]

You need to find the largest value in each row of a binary tree.

Example: Input:

1 / \ 3 2 / \ \ 5 3 9

Output: [1, 3, 9]

题目大意：找二叉树每一层最大的数～然后返回一个数组表示每一层的最大的数～

分析：先用广度优先搜索对二叉树进行层序遍历，每一层设立maxn保存每一层的最大值，然后在每一层遍历完毕之后将maxn的值放入result数组中～

```
class Solution {
public:
    vector<int> largestValues(TreeNode* root) {
        vector<int> result;
        queue<TreeNode *> q;
        if (root == NULL) return result;
        q.push(root);
        TreeNode *temp;
        while (!q.empty()) {
            int size = q.size();
            int maxn = INT_MIN;
            while (size--) {
                temp = q.front();
                q.pop();
                maxn = max(maxn, temp->val);
                if (temp->left != NULL) q.push(temp->left);
                if (temp->right != NULL) q.push(temp->right);
            }
            result.push_back(maxn);
        }
        return result;
    }
};
```

516. Longest Palindromic Subsequence [Medium]

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

Example 1: Input:

"bbbab" Output: 4 One possible longest palindromic subsequence is "bbbb". Example 2: Input:

"cbbd" Output: 2 One possible longest palindromic subsequence is "bb".

题目大意：给一个字符串s，找最长回文子串～返回这个最长回文子串的长度～

分析：使用动态规划解决～设立一个len行len列的dp数组～dp[i][j]表示字符串i～j下标所构成的子串中最长回文子串的长度～最后我们需要返回的是dp[0][len-1]的值～

dp数组这样更新：首先i指针从尾到头遍历，j指针从i指针后面一个元素开始一直遍历到尾部～一开始dp[i][i]的值都为1，如果当前i和j所指元素相等，说明能够加到i～j的回文子串的长度中，所以更新dp[i][j] = dp[i+1][j-1] + 2; 如果当前元素不相等，那么说明这两个i、j所指元素对回文串无贡献，则dp[i][j]就是从dp[i+1][j]和dp[i][j-1]中选取较大的一个值即可～

```
class Solution {
public:
    int longestPalindromeSubseq(string s) {
        int len = s.length();
        vector<vector<int>> dp(len, vector<int>(len));
        for (int i = len - 1; i >= 0; i--) {
            dp[i][i] = 1;
            for (int j = i + 1; j < len; j++) {
                if (s[i] == s[j])
                    dp[i][j] = dp[i+1][j-1] + 2;
                else
                    dp[i][j] = max(dp[i+1][j], dp[i][j-1]);
            }
        }
        return dp[0][len-1];
    }

};
```

521. Longest Uncommon Subsequence I [Easy]

Given a group of two strings, you need to find the longest uncommon subsequence of this group of two strings. The longest uncommon subsequence is defined as the longest subsequence of one of these strings and this subsequence should not be any subsequence of the other strings.

A subsequence is a sequence that can be derived from one sequence by deleting some characters without changing the order of the remaining elements. Trivially, any string is a subsequence of itself and an empty string is a subsequence of any string.

The input will be two strings, and the output needs to be the length of the longest uncommon subsequence. If the longest uncommon subsequence doesn't exist, return -1.

Example 1: Input: "aba", "cdc" Output: 3 Explanation: The longest uncommon subsequence is "aba" (or "cdc"), because "aba" is a subsequence of "aba", but not a subsequence of any other strings in the group of two strings. Note:

Both strings' lengths will not exceed 100. Only letters from a ~ z will appear in input strings.

题目大意：给定两个字符串，找这两个字符串中最长非公共子序列。 最长的非公共子序列是指这些字符串之一的最长的子序列，并且这个子序列不是其他字符串的任何子序列。

分析：如果a和b相等，那么a的任意一个子串都是b的子串，反之同理，所以a和b相等时返回-1；如果a和b不相等，返回a和b长度中较长的数字，因为只要取a和b中长度较长的那个字符串，必然是另一方的最长非公共子串～

```
class Solution {
public:
    int findLUSlength(string a, string b) {
        return a == b ? -1 : max(a.length(), b.length());
    }
};
```

532. K-diff Pairs in an Array [Easy]

Given an array of integers and an integer k, you need to find the number of unique k-diff pairs in the array. Here a k-diff pair is defined as an integer pair (i, j), where i and j are both numbers in the array and their absolute difference is k.

Example 1: Input: [3, 1, 4, 1, 5], k = 2 Output: 2 Explanation: There are two 2-diff pairs in the array, (1, 3) and (3, 5). Although we have two 1s in the input, we should only return the number of unique pairs. Example 2: Input:[1, 2, 3, 4, 5], k = 1 Output: 4 Explanation: There are four 1-diff pairs in the array, (1, 2), (2, 3), (3, 4) and (4, 5). Example 3: Input: [1, 3, 1, 5, 4], k = 0 Output: 1 Explanation: There is one 0-diff pair in the array, (1, 1). Note: The pairs (i, j) and (j, i) count as the same pair. The length of the array wont exceed 10,000. All the integers in the given input belong to the range: [-1e7, 1e7].

题目大意：给定一个整数和一个整数k的数组，你需要找到数组中唯一的k-diff对的个数。 这里k-diff对被定义为一个整数对（i，j），其中i和j都是数组中的数字，它们的绝对差值是k。

分析：对于当前遍历的nums[i]，在它前面找是否存在满足与它相差k的数字，如果找到了就将这对数字(i, j)中较大的一个数字放到set里，然后将这个数在map中标记为true表示存在这个数字等待它的后面的数字判断它是否存在（这样可以避免重复，也就是说每个数字只在它前面找是否有符合条件的数字），最后返回set的大小～

```
class Solution {
public:
    int findPairs(vector<int>& nums, int k) {
        if (k < 0) return 0;
        unordered_map<int, bool> m;
        unordered_set<int> s;
        for (int i = 0; i < nums.size(); i++) {
            if (m[nums[i] - k]) s.insert(nums[i]);
            if (m[nums[i] + k]) s.insert(nums[i] + k);
            m[nums[i]] = true;
        }
        return s.size();
    }
```

```
};
```

537. Complex Number Multiplication [Medium]

Given two strings representing two complex numbers.

You need to return a string representing their multiplication. Note i2 = -1 according to the definition.

Example 1: Input: “1+1i”, “1+1i” Output: “0+2i” Explanation: (1 + i) * (1 + i) = 1 + i2 + 2 * i = 2i, and you need convert it to the form of 0+2i. Example 2: Input: “1+-1i”, “1+-1i” Output: “0+-2i” Explanation: (1 – i) * (1 – i) = 1 + i2 – 2 * i = -2i, and you need convert it to the form of 0+-2i. Note:

The input strings will not have extra blank. The input strings will be given in the form of a+bi, where the integer a and b will both belong to the range of [-100, 100]. And the output should be also in this form.

题目大意：给两个复数，求两个数的乘积～

分析：使用sscanf和ssprinf，(m+ni)*(p+qi)=(m*p-n*q)*(n*p+m*q) i

```
class Solution {
public:
    string complexNumberMultiply(string a, string b) {
        char t[200];
        int m, n, p, q;
        sscanf(a.c_str(), "%d+%di", &m, &n);
        sscanf(b.c_str(), "%d+%di", &p, &q);
        sprintf(t, "%d+%di", (m*p-n*q), (n*p+m*q));
        string ans = t;
        return ans;
    }
};
```

538. Convert BST to Greater Tree [Easy]

Given a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus sum of all keys greater than the original key in BST.

Example:

Input: The root of a Binary Search Tree like this: 5 / \ 2 13

Output: The root of a Greater Tree like this: 18 / \ 20 13

题目大意：给定一个二叉搜索树（BST），将其转换为一个Greater Tree，使得原始BST的每个结点的键值被改变为原始键加上所有比BST中的原始键大的键的总和。

分析：因为BST的中序遍历是从小到大排列，那么BST的右根左遍历方式得到的就是从大到小的排列，遍历过程中对当前结点累计到sum中，并将sum的值赋值给当前结点，最后返回这棵树即可～

```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* convertBST(TreeNode* root) {
        dfs(root);
        return root;
    }
private:
    int sum = 0;
    void dfs(TreeNode* root) {
        if (root == NULL) return;
        if (root->right != NULL) dfs(root->right);
        sum += root->val;
        root->val = sum;
        if (root->left != NULL) dfs(root->left);
    }
};
```

541. Reverse String II [Easy]

Given a string and an integer k, you need to reverse the first k characters for every 2k characters counting from the start of the string. If there are less than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and left the other as original. Example: Input: s = “abcdefg”, k = 2 Output: “bacdfeg” Restrictions: The string consists of lower English letters only. Length of the given string and k will in the range [1, 10000]

题目大意：给一个字符串s和一个整数k，每2k长度倒置前k个字符串，如果最后剩余的长度小于k则全都倒置，否则如果剩余的长度大于k小于2k，倒置前k个，返回倒置后的字符串～

分析：遍历字符串，步长为2k，每次倒置s.begin() + i～s.begin() + i + k的字符串，如果i + k > s.length()就倒置s.begin() + i～s.begin() + s.length()即可～O(∩_∩)O～

```cpp
class Solution {
public:
    string reverseStr(string s, int k) {
        for (int i = 0; i < s.length(); i+=2*k) {
            int t = min((i + k), (int)s.length());
            reverse(s.begin() + i, s.begin() + t);
        }
        return s;
    }
};
```

543. Diameter of Binary Tree [Easy]

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

Example: Given a binary tree 1 / \ 2 3 / \ 4 5 Return 3, which is the length of the path [4,2,1,3] or [5,2,1,3].

Note: The length of path between two nodes is represented by the number of edges between them.

题目大意：给一个二叉树，计算出任意两个节点中最长的长度并返回结果～

分析：计算每个节点的深度，并在dfs过程中将每个节点左边深度+右边深度的值的最大的值保存在ans中返回～

```cpp
class Solution {
public:
    int diameterOfBinaryTree(TreeNode* root) {
        dfs(root);
        return ans;
    }
private:
    int ans = 0;
    int dfs(TreeNode* root) {
        if (root == NULL) return 0;
        int l = dfs(root->left), r = dfs(root->right);
        ans = max(ans, l + r);
        return max(l, r) + 1;
    }
};
```

547. Friend Circles [Medium]

There are N students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a direct friend of B, and B is a direct friend of C, then A is an indirect friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a N*N matrix M representing the friend relationship between students in the class. If M[i][j] = 1, then the ith and jth students are direct friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

Example 1: Input: [[1,1,0], [1,1,0], [0,0,1]] Output: 2 Explanation:The 0th and 1st students are direct friends, so they are in a friend circle. The 2nd student himself is in a friend circle. So return 2. Example 2: Input: [[1,1,0], [1,1,1], [0,1,1]] Output: 1 Explanation:The 0th and 1st students are direct friends, the 1st and 2nd students are direct friends, so the 0th and 2nd students are indirect friends. All of them are in the same friend circle, so return 1. Note: N is in range [1,200]. M[i][i] = 1 for all students. If M[i][j] = 1, then M[j][i] = 1.

题目大意：班上有N个学生。 有些是朋友，有些则不是。 他们的友谊本质上是传递性的。 例如，如果A是B的直接朋友，B是C的直接朋友，那么A是C的间接朋友。我们定义了一个朋友圈是一群直接或间接的朋友。给定N * N矩阵M代表班级中学生之间的朋友关系。 如果M[i][j] = 1，那么第i和第j个学生是彼此直接的朋友，否则不是，输出所有学生中的朋友圈的总数

分析：用并查集，cnt一开始为n，每个人的父亲也都是自己，将每一对朋友的父亲结点找到，如果他们的父亲结点不是同一个，那么就合并为一个集合，并将cnt-1，最后输出cnt

```
class Solution {
public:
    int findCircleNum(vector<vector<int>>& M) {
        int n = M.size(), cnt = M.size();
        father.resize(n);
        for (int i = 0; i < n; i++) father[i] = i;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (M[i][j] == 1) {
                    int faA = findFather(i);
                    int faB = findFather(j);
                    if (faA != faB) {
                        father[faA] = father[faB];
                        cnt--;
                    }
                }

            }
        }
        return cnt;
    }
private:
    vector<int> father;
    int findFather(int x) {
```

```
        return x == father[x] ? x : findFather(father[x]);
    }
};
```

551. Student Attendance Record I [Easy]

You are given a string representing an attendance record for a student. The record only contains the following three characters: 'A' : Absent. 'L' : Late. 'P' : Present. A student could be rewarded if his attendance record doesn't contain more than one 'A' (absent) or more than two continuous 'L' (late).

You need to return whether the student could be rewarded according to his attendance record.

Example 1: Input: "PPALLP" Output: True Example 2: Input: "PPALLL" Output: False

分析：正则表达式匹配，如果出现三次连续的LLL或者两次AA则返回false

```
class Solution {
public:
    bool checkRecord(string s) {
        return !regex_match(s, regex(".*LLL.*|.*A.*A.*"));
    }
};
```

553. Optimal Division [Medium]

Given a list of positive integers, the adjacent integers will perform the float division. For example, [2,3,4] -> 2 / 3 / 4.

However, you can add any number of parenthesis at any position to change the priority of operations. You should find out how to add parenthesis to get the maximum result, and return the corresponding expression in string format. Your expression should NOT contain redundant parenthesis.

Example: Input: [1000,100,10,2] Output: "1000/(100/10/2)" Explanation: 1000/(100/10/2) = 1000/((100/10)/2) = 200 However, the bold parenthesis in "1000/((100/10)/2)" are redundant, since they don't influence the operation priority. So you should return "1000/(100/10/2)".

Other cases: 1000/(100/10)/2 = 50 1000/(100/(10/2)) = 50 1000/100/10/2 = 0.5 1000/100/(10/2) = 2 Note:

The length of the input array is [1, 10]. Elements in the given array will be in range [2, 1000]. There is only one optimal division for each test case.

题目大意：给定一个正整数列表，相邻的整数将执行浮点除法。 例如[2,3,4] – > 2/3/4。但是，您可以在任何位置添加任意数量的括号以更改操作的优先级。您应该找到如何添加括号以获得最大结果，并以字符串格式返回相应的表达式，你的表达不应该包含多余的括号。

分析：要想得到最大的结果，只要使除数尽可能大，被除数尽可能小。被除过的数一定会变得更小，所以括号加在第一个数后面，括号内的数按从前到后顺序（不用添加冗余的括号）即可～

```cpp
class Solution {
public:
    string optimalDivision(vector<int>& nums) {
        string ans = to_string(nums[0]);
        if (nums.size() == 1) return ans;
        if (nums.size() == 2) return ans + "/" + to_string(nums[1]);
        ans = ans + "/(" + to_string(nums[1]);
        for (int i = 2; i < nums.size(); i++) {
            ans = ans + "/" + to_string(nums[i]);
        }
        return ans + ")";
    }
};
```

561. Array Partition I [Easy]

Given an array of 2n integers, your task is to group these integers into n pairs of integer, say (a1, b1), (a2, b2), ..., (an, bn) which makes sum of min(ai, bi) for all i from 1 to n as large as possible.

Example 1: Input: [1,4,3,2]

Output: 4 Explanation: n is 2, and the maximum sum of pairs is 4 = min(1, 2) + min(3, 4). Note: n is a positive integer, which is in the range of [1, 10000]. All the integers in the array will be in the range of [-10000, 10000].

题目大意：给2n个数，请把数字分为2个一组，问所有组（取每组数的较小的那一数字）累加的和最大为多少～

分析：把数组从小到大排列，第1个、第3个、...第2n-1个数字之和即为所求～

```cpp
class Solution {
public:
    int arrayPairSum(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int ans = 0;
        for (int i = 0; i < nums.size(); i+=2)
            ans += nums[i];
        return ans;
    }
};
```

563. Binary Tree Tilt [Easy]

Given a binary tree, return the tilt of the whole tree.

The tilt of a tree node is defined as the absolute difference between the sum of all left subtree node values and the sum of all right subtree node values. Null node has tilt 0.

The tilt of the whole tree is defined as the sum of all nodes' tilt.

Example: Input: 1 / \ 2 3 Output: 1 Explanation: Tilt of node 2 : 0 Tilt of node 3 : 0 Tilt of node 1 : |2-3| = 1 Tilt of binary tree : 0 + 0 + 1 = 1 Note:

The sum of node values in any subtree won't exceed the range of 32-bit integer. All the tilt values won't exceed the range of 32-bit integer.

题目大意：给定一棵二叉树，返回整棵树的倾斜度。树节点的倾斜度被定义为所有左子树节点值的总和与所有右节点值的总和之间的绝对差值。 空节点具有倾斜0。

分析：getSum函数用来计算以当前结点为根的子树的结点值总和，dfs用来遍历树，对于每个节点计算getSum的左子树和右子树之差累加，最后返回累加值～

```
class Solution {
public:
    int findTilt(TreeNode* root) {
        dfs(root);
        return ans;
    }
private:
    int ans = 0;
    int getSum(TreeNode* root) {
        if (root == NULL) return 0;
        int l = getSum(root->left), r = getSum(root->right);
        return l + r + root->val;
    }
    void dfs(TreeNode* root) {
        if (root == NULL) return;
        ans += abs(getSum(root->left)-getSum(root->right));
        dfs(root->left);
        dfs(root->right);
    }
};
```

566. Reshape the Matrix [Easy]

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.

You're given a matrix represented by a two-dimensional array, and two positive integers r and c representing the row number and column number of the wanted reshaped matrix, respectively.

The reshaped matrix need to be filled with all the elements of the original matrix in the same row-traversing order as they were.

If the ‘reshape operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

Example 1: Input: nums = [[1,2], [3,4]] r = 1, c = 4 Output: [[1,2,3,4]] Explanation: The row-traversing of nums is [1,2,3,4]. The new reshaped matrix is a 1 * 4 matrix, fill it row by row by using the previous list. Example 2: Input: nums = [[1,2], [3,4]] r = 2, c = 4 Output: [[1,2], [3,4]] Explanation: There is no way to reshape a 2 * 2 matrix to a 2 * 4 matrix. So output the original matrix. Note: The height and width of the given matrix is in range [1, 100]. The given r and c are all positive.

题目大意：给出一个由二维数组表示的矩阵，两个正整数r和c分别代表所需的整形矩阵的行号和列号。重构后的矩阵需要以原始矩阵的所有元素按照相同的行遍数顺序填充。如果“给定参数的重塑操作是可能的和合法的，则输出新的重塑矩阵; 否则，输出原始矩阵。

分析：如果原nums的行列乘积等于r*c则返回nums，否则建立r行c列的数组，将ans[i/c][i%c] = nums[i/m][i%m]即可～

```
class Solution {
public:
    vector<vector<int>> matrixReshape(vector<vector<int>>& nums, int r, int c) {
        int n = nums.size(), m = nums[0].size();
        if (n * m != r * c) return nums;
        vector<vector<int>> ans(r, vector<int>(c));
        for (int i = 0; i < r * c; i++) ans[i/c][i%c] = nums[i/m][i%m];
        return ans;
    }
};
```

572. Subtree of Another Tree [Easy]

Given two non-empty binary trees s and t, check whether tree t has exactly the same structure and node values with a subtree of s. A subtree of s is a tree consists of a node in s and all of this node’s descendants. The tree s could also be considered as a subtree of itself.

Example 1: Given tree s:

3 / \ 4 5 / \ 1 2 Given tree t: 4 / \ 1 2 Return true, because t has the same structure and node values with a subtree of s. Example 2: Given tree s:

3 / \ 4 5 / \ 1 2 / 0 Given tree t: 4 / \ 1 2 Return false.

题目大意：给定两个非空的二叉树s和t，检查树t是否具有与s的子树完全相同的结构和节点值。 s的子树是由s中的一个节点和所有这个节点的后代组成的一棵树。 树也可以被认为是它自己的一个子树。

分析：isSame判断两棵树是否相等，isSubtree中如果s->val == t->val && isSame(s, t)则返回true表示以s为根结点和以t为根结点的树相等，否则判断s->left和t是否相等，s->right和t是否相等，只要有一个相等即可返回true～

```cpp
class Solution {
public:
    bool isSubtree(TreeNode* s, TreeNode* t) {
        if (s == NULL && t == NULL) return true;
        if (s == NULL || t == NULL) return false;
        if (s->val == t->val && isSame(s, t)) return true;
        return isSubtree(s->left, t) || isSubtree(s->right, t);
    }
private:
    bool isSame(TreeNode* r, TreeNode* t) {
        if (r == NULL && t == NULL) return true;
        if (r == NULL || t == NULL || r->val != t->val) return false;
        return (isSame(r->left, t->left) && isSame(r->right, t->right));
    }
};
```

575. Distribute Candies [Easy]

Given an integer array with even length, where different numbers in this array represent different kinds of candies. Each number means one candy of the corresponding kind. You need to distribute these candies equally in number to brother and sister. Return the maximum number of kinds of candies the sister could gain. Example 1: Input: candies = [1,1,2,2,3,3] Output: 3 Explanation: There are three different kinds of candies (1, 2 and 3), and two candies for each kind. Optimal distribution: The sister has candies [1,2,3] and the brother has candies [1,2,3], too. The sister has three different kinds of candies. Example 2: Input: candies = [1,1,2,3] Output: 2 Explanation: For example, the sister has candies [2,3] and the brother has candies [1,1]. The sister has two different kinds of candies, the brother has only one kind of candies. Note:

The length of the given array is in range [2, 10,000], and will be even. The number in given array is in range [-100,000, 100,000].

题目大意：给定一个长度为偶数的整型数组，这个数组中的不同数字表示不同类型的糖果。 每个数字表示相应种类的一个糖果。你需要把这些糖果平均分配给弟弟和姐姐。 返回姐姐可以获得的最大数量的糖果种类。

分析：放入unorder_set的集合中，集合中元素的个数就是糖果的种类，如果种类大于一半则返回一半的值，否则返回集合内的元素种类即可～

```cpp
class Solution {
public:
    int distributeCandies(vector<int>& candies) {
        unordered_set<int> s;
        for (int i = 0; i < candies.size(); i++) s.insert(candies[i]);
        return min(s.size(), candies.size() / 2);
    }
};
```

581. Shortest Unsorted Continuous Subarray [Easy]

Given an integer array, you need to find one continuous subarray that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order, too.

You need to find the shortest such subarray and output its length.

Example 1: Input: [2, 6, 4, 8, 10, 9, 15] Output: 5 Explanation: You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order. Note: Then length of the input array is in range [1, 10,000]. The input array may contain duplicates, so ascending order here means <=.

题目大意：给定一个整数数组，你需要找到一个连续的子数组，如果你按升序对这个子数组排序，那么整个数组将也是升序。你需要找到最短的这样的子数组并输出它的长度。

分析：对v排序，i找到第一个和排序后的数组v不相等的元素，j找到最后一个和排序后的数组v不相等的元素，如果i<=j说明存在这样一个长度，长度为j-i+1，否则不存在这样一个长度，则返回0～

```cpp
class Solution {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        vector<int> v(nums);
        sort(v.begin(), v.end());
        int i = 0, j = nums.size() - 1;
        while (i < nums.size() && nums[i] == v[i]) i++;
        while (j >= 0 && nums[j] == v[j]) j--;
        return i <= j ? j - i + 1 : 0;
    }
};
```

599. Minimum Index Sum of Two Lists [Easy]

Suppose Andy and Doris want to choose a restaurant for dinner, and they both have a list of favorite restaurants represented by strings.

You need to help them find out their common interest with the least list index sum. If there is a choice tie between answers, output all of them with no order requirement. You could assume there always exists an answer.

Example 1: Input: [“Shogun”, “Tapioca Express”, “Burger King”, “KFC”] [“Piatti”, “The Grill at Torrey Pines”, “Hungry Hunter Steakhouse”, “Shogun”] Output: [“Shogun”] Explanation: The only restaurant they both like is “Shogun”. Example 2: Input: [“Shogun”, “Tapioca Express”, “Burger King”, “KFC”] [“KFC”, “Shogun”, “Burger King”] Output: [“Shogun”] Explanation: The restaurant they both like and have the least index sum is “Shogun” with index sum 1 (0+1). Note: The length of both lists will be in the range of [1, 1000]. The length of strings in both lists will be in the range of [1, 30]. The index is starting from 0 to the list length minus 1. No duplicates in both lists.

题目大意：假设安迪和多丽丝想选择一家餐厅吃饭，他们有一串最喜欢的餐厅名单。你需要帮助他们用最小的指数总和找到他们的共同兴趣。 如果答案有多个则没有顺序要求地返回所有答案在string数组中～

分析：当i+j比minSum小时将数组清空，minSum赋值为i+j，list1[i]放入ans数组中

```
class Solution {
public:
    vector<string> findRestaurant(vector<string>& list1,
vector<string>& list2) {
        vector<string> ans;
        int minSum = 2000;
        for (int i = 0; i < list1.size(); i++) {
            for (int j = 0; j < list2.size(); j++) {
                if (list1[i] == list2[j] && minSum > i + j) {
                    ans.clear();
                    ans.push_back(list1[i]);
                    minSum = i + j;
                } else if (list1[i] == list2[j] && minSum == i + j) {
                    ans.push_back(list1[i]);
                }
            }
        }
        return ans;
    }
};
```

628. Maximum Product of Three Numbers [Easy]

Given an integer array, find three numbers whose product is maximum and output the maximum product.

Example 1: Input: [1,2,3] Output: 6 Example 2: Input: [1,2,3,4] Output: 24 Note: The length of the given array will be in range [3,104] and all elements are in the range [-1000, 1000]. Multiplication of any three numbers in the input won’t exceed the range of 32-bit signed integer.

题目大意：给一个数组，找三个数字，使这三个数乘积最大，返回乘积的最大值～

分析：先对数组排序，要么是最后三个数字乘积，要么可能会有负数，则是前两个数的乘积*最后一个数字的乘积，取这个两个乘积结果的最大值即可～

```
class Solution {
public:
    int maximumProduct(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        return max(nums[0] * nums[1] * nums[n-1], nums[n-3] * nums[n-2] * nums[n-1]);
    }
};
```

633. Sum of Square Numbers [Easy]

Given a non-negative integer c, your task is to decide whether there're two integers a and b such that a2 + b2 = c.

Example 1: Input: 5 Output: True Explanation: 1 * 1 + 2 * 2 = 5 Example 2: Input: 3 Output: False

题目大意：给一个数c，判断是否存在两个数字a b满足a*a+b*b=c，存在就返回true，否则返回false

分析：i和j分别从根号c开始到0，如果找到了i * i + j * j == c就return true，如果i * i + j * j < c就break第二层for循环，最后如果都没有找到就返回false～

```
class Solution {
public:
    bool judgeSquareSum(int c) {
        int t = sqrt(c);
        for (int i = t; i >= 0; i--) {
            for (int j = t; j >= 0; j--) {
                if (i * i + j * j == c) return true;
                if (i * i + j * j < c) break;
            }
        }
        return false;
    }
};
```

637. Average of Levels in Binary Tree [Easy]

Given a non-empty binary tree, return the average value of the nodes on each level in the form of an array. Example 1: Input: 3 / \ 9 20 / \ 15 7 Output: [3, 14.5, 11] Explanation: The average value of nodes on level 0 is 3, on level 1 is 14.5, and on level 2 is 11. Hence return [3, 14.5, 11]. Note: The range of node's value is in the range of 32-bit signed integer.

题目大意：计算一棵树的每一层结点的值的平均值。

```cpp
class Solution {
public:
    vector<double> averageOfLevels(TreeNode* root) {
        dfs(root, 0);
        vector<double> res;
        for(int i = 0; i <= ma ;i++) res.push_back(1.0*sum[i]/cnt[i]);
        return res;
    }
private:
    int ma = 0;
    long long sum[1000], cnt[1000];
    void dfs(TreeNode* root, int lev) {
        if(root == NULL) return;
        sum[lev] += root->val;
        ma = max(ma,lev);
        cnt[lev]++;
        dfs(root->left, lev + 1);
        dfs(root->right, lev + 1);
    }
};
```

643. Maximum Average Subarray I [Easy]

Given an array consisting of n integers, find the contiguous subarray of given length k that has the maximum average value. And you need to output the maximum average value.

Example 1: Input: [1,12,-5,-6,50,3], k = 4 Output: 12.75 Explanation: Maximum average is (12-5-6+50)/4 = 51/4 = 12.75 Note: 1 <= k <= n <= 30,000. Elements of the given array will be in the range [-10,000, 10,000].

题目大意：给定一个由n个整数组成的数组，找到具有最大平均值的给定长度k的连续子数组。你需要输出最大的平均值～

分析：sum[i]保存0～i个数字的和，每次将(sum[i]-sum[i-k])的最大值保存在ans中，然后返回ans*1.0/k

```cpp
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k) {
        if (nums.size() == 0) return 0.0;
        vector<int> sum(nums.size());
        sum[0] = nums[0];
        for (int i = 1; i < k; i++) sum[i] = sum[i-1] + nums[i];
        int ans = sum[k-1];
        for (int i = k; i < nums.size(); i++) {
            sum[i] = sum[i-1] + nums[i];
```

```
            ans = max(ans, sum[i] - sum[i-k]);
        }
        return ans * 1.0 / k;
    }
};
```

650. 2 Keys Keyboard [Medium]

Initially on a notepad only one character 'A' is present. You can perform two operations on this notepad for each step:

Copy All: You can copy all the characters present on the notepad (partial copy is not allowed). Paste: You can paste the characters which are copied last time. Given a number n. You have to get exactly n 'A' on the notepad by performing the minimum number of steps permitted. Output the minimum number of steps to get n 'A'.

Example 1: Input: 3 Output: 3 Explanation: Intitally, we have one character 'A'. In step 1, we use Copy All operation. In step 2, we use Paste operation to get 'AA'. In step 3, we use Paste operation to get 'AAA'. Note: The n will be in the range [1, 1000].

题目大意：一开始给一个A，每次只能复制所有、或者粘贴上一次复制的内容。问要想最后变成n个A，至少得经过多少步～

分析：可以用贪心算法解决，i从2到√n，尝试是否能被n整除，如果能被整除，说明可以通过复制1次粘贴i-1次得到，则计数器ans加上i次，然后将n除以i。再次判断是否能被i整除，直至不能整除为止。然后尝试下一个i……最终n如果等于1则直接返回ans，否则要加上n表示对A复制一次粘贴n-1次～

```
class Solution {
public:
    int minSteps(int n) {
        int ans = 0;
        for (int i = 2; i <= sqrt(n); i++) {
            while (n % i == 0) {
                ans += i;
                n /= i;
            }
        }
        if (n != 1) ans += n;
        return ans;
    }
};
```

653. Two Sum IV – Input is a BST [Easy]

Given a Binary Search Tree and a target number, return true if there exist two elements in the BST such that their sum is equal to the given target.

Example 1: Input: 5 / \ 3 6 / \ \ 2 4 7

Target = 9

Output: True Example 2: Input: 5 / \ 3 6 / \ \ 2 4 7

Target = 28

Output: False

题目大意：给一棵二叉搜索树和一个目标数字，如果这棵树中存在两个元素，这两个元素的和等于目标数字，则返回true

分析：设立set，如果set里面存在k – 当前结点的值，则返回true；每次将当前结点的值放入set中，否则返回它左子树的结果 || 右子树的结果。相当于每次比较的是当前结点i之前的所有数字中是否有和i相加为k的数字，如果有就返回true，否则就继续遍历左子树和右子树，只要其中一个满足就返回true

```
/**
 * 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:
    set<int> s;
    bool findTarget(TreeNode* root, int k) {
        if (root == NULL) return false;
        if (s.count(k - root->val)) return true;
        s.insert(root->val);
        return findTarget(root->left, k) || findTarget(root->right, k);
    }
};
```

654. Maximum Binary Tree [Medium]

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

The root is the maximum number in the array. The left subtree is the maximum tree constructed from left part subarray divided by the maximum number. The right subtree is the maximum tree constructed from right part subarray divided by the maximum number. Construct the maximum tree by the given array and output the root node of this tree.

Example 1: Input: [3,2,1,6,0,5] Output: return the tree root node representing the following tree:

6 / \ 3 5 \ / 2 0 \ 1 Note: The size of the given array will be in the range [1,1000].

分析：用递归，l和r分别表示使用在nums[l]到nums[r]区间内的数字进行建树，每次通过idx找到l到r之间的最大值对应的下标idx，使用nums[idx]值new一个新TreeNode结点root，然后将root的左边用l～idx-1建树，右边用idx+1～r建树，然后返回root结点即完成了建树～

```
/**
 * 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* constructMaximumBinaryTree(vector<int>& nums) {
        return build(0, nums.size()-1, nums);
    }
private:
    int n;
    TreeNode* build(int l, int r, vector<int>& nums) {
        if (l > r) return NULL;
        int idx = l;
        for (int i = l + 1; i <= r; i++) {
            if (nums[i] > nums[idx]) idx = i;
        }
        TreeNode* root = new TreeNode(nums[idx]);
        root->left = build(l, idx-1, nums);
        root->right = build(idx+1, r, nums);
        return root;
    }
};
```

655. Print Binary Tree [Medium]

Print a binary tree in an m*n 2D string array following these rules:

The row number m should be equal to the height of the given binary tree. The column number n should always be an odd number. The root node’s value (in string format) should be put in the exactly middle of the first row it can be put. The column and the row where the root node belongs will separate the rest space into two parts (left-bottom part and right-bottom part). You should print the left subtree in the left-bottom part and print the right subtree in the right-bottom part. The left-bottom part and the right-bottom part should have the same size. Even if one subtree is none while the other is not, you don’t need to print anything for the none subtree but still need to leave the space as large as that for the other subtree. However, if two subtrees are none, then you don’t need to leave space for both of them. Each unused space should contain an empty string “”. Print the subtrees following the same rules. Example 1: Input:

1 / 2 Output: [[“”, “1”, “”], [“2”, “”, “”]] Example 2: Input: 1 / \ 2 3 \ 4 Output: [[“”, “”, “”, “1”, “”, “”, “”], [“”, “2”, “”, “”, “”, “3”, “”], [“”, “”, “4”, “”, “”, “”, “”]] Example 3: Input: 1 / \ 2 5 / 3 / 4 Output:

[[“”, “”, “”, “”, “”, “”, “”, “1”, “”, “”, “”, “”, “”, “”, “”, “”] [“”, “”, “”, “2”, “”, “”, “”, “”, “”, “”, “”, “5”, “”, “”, “”] [“”, “3”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”] [“4”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”, “”]] Note: The height of binary tree is in the range of [1, 10].

题目大意：按照以下规则在一个m * n的二维字符串数组中打印一个二叉树： 行号m应该等于给定二叉树的高度。 列号n应始终为奇数。 根节点的值（以字符串格式）应该放在第一行的正中间。 根节点所属的行和列将剩余空间分成两部分（左下部分和右下部分）。 您应该在左下角打印左侧子树，并在右下侧打印右侧子树。 左下部分和右下部分应该具有相同的尺寸。 即使一个子树不存在，而另一个子树不存在，您也不需要为无子树打印任何东西，但仍然需要留下与另一子树相同的空间。 但是，如果两个子树都没有，那么你就不需要为它们留下空间。 每个未使用的空间应该包含一个空字符串“”。 按照相同的规则打印子树。

分析：首先计算出树的高度h，然后建立h行、(2的h次方-1)列的字符串数组，进行先序遍历，设立l和r表示当前填充的子树对应的字符串数组的左右下标区域，h表示当前的层数，每次将root->val的值存储入ans[h][mid]中，其中mid=(l+r)/2，然后继续遍历左子树和右子树，将区间分为0～mid-1和mid+1～r，高度h加1，直至遍历完成后返回ans数组～

```
/**
 * 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<string>> printTree(TreeNode* root) {
        int h = height(root);
        int c = pow(2, h) - 1;
        vector<vector<string>> ans(h, vector<string>(c));
        dfs(root, 0, c-1, 0, ans);
        return ans;
    }
private:
    int height(TreeNode* root) {
        if (root == NULL) return 0;
        return max(height(root->left), height(root->right)) + 1;
    }
    void dfs(TreeNode* root, int l, int r, int h,
vector<vector<string>>& ans) {
        if (root == NULL) return;
        if (l > r) return;
        int mid = (l + r) / 2;
```

```
        ans[h][mid] = to_string(root->val);
        dfs(root->left, l, mid-1, h+1, ans);
        dfs(root->right, mid+1, r, h+1, ans);
    }
};
```

657. Judge Route Circle [Easy]

Initially, there is a Robot at position (0, 0). Given a sequence of its moves, judge if this robot makes a circle, which means it moves back to the original place.

The move sequence is represented by a string. And each move is represent by a character. The valid robot moves are R (Right), L (Left), U (Up) and D (down). The output should be true or false representing whether the robot makes a circle.

Example 1: Input: “UD” Output: true Example 2: Input: “LL” Output: false

题目大意：最初，位置（0，0）处有一个机器人。 给出它的一系列动作，判断这个机器人是否有一个圆圈，这意味着它回到原来的位置。移动顺序由一个字符串表示。 而每一个动作都是由一个人物来表现的。 有效的机器人移动R（右），L（左），U（上）和D（下）。 输出应该是真或假，表示机器人是否成圈。

分析：计算UDLR出现的次数保存在map中，如果U和D出现的次数相同且L和R出现的次数相同则为true～

```
class Solution {
public:
    bool judgeCircle(string moves) {
        map<char, int> m;
        for(int i = 0; i < moves.length(); i++) m[moves[i]]++;
        return (m['L'] == m['R'] && m['U'] == m['D']);
    }
};
```

671. Second Minimum Node In a Binary Tree [Easy]

Given a non-empty special binary tree consisting of nodes with the non-negative value, where each node in this tree has exactly two or zero sub-node. If the node has two sub-nodes, then this node’s value is the smaller value among its two sub-nodes.

Given such a binary tree, you need to output the second minimum value in the set made of all the nodes’ value in the whole tree.

If no such second minimum value exists, output -1 instead.

Example 1: Input: 2 / \ 2 5 / \ 5 7

Output: 5 Explanation: The smallest value is 2, the second smallest value is 5. Example 2: Input: 2 / \ 2 2

Output: -1 Explanation: The smallest value is 2, but there is not any second smallest value.

题目大意：给一个非空的树，这个树所有值非负，而且每个结点只有2个孩子或者0个孩子，如果当前结点有两个孩子，则这个结点的值比它的两个孩子的值都小，要求输出这个树第二小的值。

分析：遍历树，将所有的值放入集合中，输出集合的第二个数；如果集合的大小小于等于1，则输出-1

```
/**
 * 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 findSecondMinimumValue(TreeNode* root) {
        dfs(root);
        return s.size() <= 1 ? -1 : (*++s.begin());
    }
private:
    set<int> s;
    void dfs(TreeNode* root) {
        if (root == NULL) return;
        if (root->left != NULL) dfs(root->left);
        s.insert(root->val);
        if (root->right != NULL) dfs(root->right);
    }
};
```

674. Longest Continuous Increasing Subsequence [Easy]

Given an unsorted array of integers, find the length of longest continuous increasing subsequence (subarray).

Example 1: Input: [1,3,5,4,7] Output: 3 Explanation: The longest continuous increasing subsequence is [1,3,5], its length is 3. Even though [1,3,5,7] is also an increasing subsequence, it is not a continuous one where 5 and 7 are separated by 4. Example 2: Input: [2,2,2,2,2] Output: 1 Explanation: The longest continuous increasing subsequence is [2], its length is 1. Note: Length of the array will not exceed 10,000.

题目大意：给一个乱序数组，返回这个数组中递增连续子序列中最长的长度～

分析：遍历从i = 1到结尾，每次比较nums[i-1]和nums[i]，如果是递增的就将temp++，否则temp=1，每次将temp最大值保存在ans中，最后返回ans的值～

```
class Solution {
public:
    int findLengthOfLCIS(vector<int>& nums) {
        int temp = 1, ans = 1;
        for (int i = 1; i < nums.size(); i++) {
            temp = (nums[i-1] < nums[i]) ? temp + 1 : 1;
            ans = max(ans, temp);
        }
        return nums.size() == 0 ? 0 : ans;
    }
};
```

695. Max Area of Island [Easy]

Given a non-empty 2D array grid of 0's and 1's, an island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

Find the maximum area of an island in the given 2D array. (If there is no island, the maximum area is 0.)

Example 1: [[0,0,1,0,0,0,0,1,0,0,0,0,0], [0,0,0,0,0,0,0,1,1,1,0,0,0], [0,1,1,0,1,0,0,0,0,0,0,0,0], [0,1,0,0,1,1,0,0,1,0,1,0,0], [0,1,0,0,1,1,0,0,1,1,1,0,0], [0,0,0,0,0,0,0,0,0,0,1,0,0], [0,0,0,0,0,0,0,1,1,1,0,0,0], [0,0,0,0,0,0,0,1,1,0,0,0,0]] Given the above grid, return 6. Note the answer is not 11, because the island must be connected 4-directionally. Example 2: [[0,0,0,0,0,0,0,0]] Given the above grid, return 0. Note: The length of each dimension in the given grid does not exceed 50.

题目大意：给一个地图，0表示水1表示陆地，计算最大的一个岛屿的面积～

分析：遍历所有grid[x][y] == 1的地盘，每次将tempcnt = 0，然后进行dfs，dfs中从4个方向遍历，每次对于grid[x][y] == 1的地盘进行更深一层的dfs，进入dfs先将当前地盘标记为0且tempcnt++，最后tempcnt的值即为这个岛屿的数量，当main函数中遍历过所有的岛屿后返回cnt的最大值即是面积最大的岛屿～

```
class Solution {
public:
    int maxAreaOfIsland(vector<vector<int>>& grid) {
        if (grid.size() == 0) return 0;
        n = grid.size(), m = grid[0].size();
        for (int x = 0; x < n; x++) {
            for (int y = 0; y < m; y++) {
                if (grid[x][y] == 1) {
                    tempcnt = 0;
                    dfs(x, y, grid);
                    cnt = max(cnt, tempcnt);
                }
            }
```

```
        }
        return cnt;
    }
private:
    int n, m, cnt = 0, tempcnt = 0;
    int arr[5] = {0, 1, 0, -1, 0};
    void dfs(int x, int y, vector<vector<int>>& grid) {
        grid[x][y] = 0;
        tempcnt++;
        for (int i = 0; i < 4; i++) {
            int tx = x + arr[i], ty = y + arr[i+1];
            if (tx >= 0 && tx < n && ty >= 0 && ty < m && grid[tx][ty]
== 1)
                dfs(tx, ty, grid);
        }
    }
};
```

717. 1-bit and 2-bit Characters [Easy]

We have two special characters. The first character can be represented by one bit 0. The second character can be represented by two bits (10 or 11).

Now given a string represented by several bits. Return whether the last character must be a one-bit character or not. The given string will always end with a zero.

Example 1: Input: bits = [1, 0, 0] Output: True Explanation: The only way to decode it is two-bit character and one-bit character. So the last character is one-bit character. Example 2: Input: bits = [1, 1, 1, 0] Output: False Explanation: The only way to decode it is two-bit character and two-bit character. So the last character is NOT one-bit character. Note:

1 <= len(bits) <= 1000. bits[i] is always 0 or 1.

题目大意：我们有两个特殊字符。 第一个字符可以用1位表示。第二个字符可以用2位（10或11）表示。现在给出一个由几位表示的字符串。 返回最后一个字符是否是一位字符。 给定的字符串将始终以0结束

分析：i从0开始遍历整个bits数组，当i遇到0时走一步，否则走2步，判断是否会走到最后一个元素～

```cpp
class Solution {
public:
    bool isOneBitCharacter(vector<int>& bits) {
        int i = 0;
        while(i < bits.size()) {
            if (i == (bits.size() - 1)) return true;
            if (bits[i] == 0) i++;
            else i += 2;
        }
        return false;
    }
};
```

724. Find Pivot Index [Easy]

Given an array of integers nums, write a method that returns the “pivot” index of this array.

We define the pivot index as the index where the sum of the numbers to the left of the index is equal to the sum of the numbers to the right of the index.

If no such index exists, we should return -1. If there are multiple pivot indexes, you should return the left-most pivot index.

Example 1: Input: nums = [1, 7, 3, 6, 5, 6] Output: 3 Explanation: The sum of the numbers to the left of index 3 (nums[3] = 6) is equal to the sum of numbers to the right of index 3. Also, 3 is the first index where this occurs. Example 2: Input: nums = [1, 2, 3] Output: -1 Explanation: There is no index that satisfies the conditions in the problem statement. Note:

The length of nums will be in the range [0, 10000]. Each element nums[i] will be an integer in the range [-1000, 1000].

题目大意：在一个数组中找到一个数字的下标，要求这个数字左边的数字和等于右边的数字和，如果不存在就返回-1

分析：计算数组所有元素和sum，然后遍历数组，每次将前i-1个数字的和累加在t中，每次从sum中减去nums[i]，这样sum就是nums[i]后面所有数字的和，如果sum == t就返回i，否则就返回-1

```cpp
class Solution {
public:
    int pivotIndex(vector<int>& nums) {
        int n = nums.size(), sum = 0, t = 0;
        if (n == 0) return -1;
        if (n == 1) return 0;
        for (int i : nums) sum += i;
        for (int i = 0; i < n; i++) {
            sum -= nums[i];
            if (sum == t) return i;
            t += nums[i];
        }
```

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

728. Self Dividing Numbers [Easy]

A self-dividing number is a number that is divisible by every digit it contains.

For example, 128 is a self-dividing number because 128 % 1 == 0, 128 % 2 == 0, and 128 % 8 == 0.

Also, a self-dividing number is not allowed to contain the digit zero.

Given a lower and upper number bound, output a list of every possible self dividing number, including the bounds if possible.

Example 1: Input: left = 1, right = 22 Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22] Note:

The boundaries of each input argument are 1 <= left <= right <= 10000.

题目大意：一个自我分裂的数字是一个数字，它可以被每个数字所包含。例如，128是一个自分数，因为128%1 == 0，128%2 == 0和128%8 == 0。而且，一个自分数的数字不允许包含数字零。给定一个较低和较高的数字边界，输出每一个可能的自分数的列表，如果可能的话，包括边界。

分析：将left和right中每一个数字i转为string，对string的每一位取余，如果当前位等于0或者取余结果不为0则flag标记为false，把所有flag为true的数字放入ans数组中返回～

```
class Solution {
public:
    vector<int> selfDividingNumbers(int left, int right) {
        vector<int> ans;
        for (int i = left; i <= right; i++) {
            string s = to_string(i);
            bool flag = true;
            for (int j = 0; j < s.length(); j++) {
                if (s[j] == '0' || i % (s[j] - '0') != 0) {
                    flag = false;
                    break;
                }
            }
            if (flag) ans.push_back(i);
        }
        return ans;
    }
};
```

733. Flood Fill [Easy]

An image is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

Given a coordinate (sr, sc) representing the starting pixel (row and column) of the flood fill, and a pixel value newColor, “flood fill” the image.

To perform a “flood fill”, consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color as the starting pixel), and so on. Replace the color of all of the aforementioned pixels with the newColor.

At the end, return the modified image.

Example 1: Input: image = [[1,1,1],[1,1,0],[1,0,1]] sr = 1, sc = 1, newColor = 2 Output: [[2,2,2],[2,2,0],[2,0,1]] Explanation: From the center of the image (with position (sr, sc) = (1, 1)), all pixels connected by a path of the same color as the starting pixel are colored with the new color. Note the bottom corner is not colored 2, because it is not 4-directionally connected to the starting pixel. Note:

The length of image and image[0] will be in the range [1, 50]. The given starting pixel will satisfy 0 <= sr < image.length and 0 <= sc < image[0].length. The value of each color in image[i][j] and newColor will be an integer in [0, 65535].

题目大意：给定表示填充的开始像素（行和列）的坐标（sr，sc）和像素值newColor，“填充”图像。从上下左右四个方向填充，只要是和原来颜色一样的都填充为新的颜色～

分析：从(sr, sc)处遍历四个方向，用visit标记是否访问过，每次将所有和原来相同颜色的坐标填充为新的颜色并将当前坐标标记为访问过，最后返回image数组～

```
class Solution {
public:
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {
        n = image.size();
        if (n == 0) return image;
        m = image[0].size();
        visit.resize(n, vector<bool>(m, false));
        dfs(image, sr, sc, image[sr][sc], newColor);
        return image;
    }
private:
    int n, m, arr[5] = {1, 0 , -1, 0, 1};
    vector<vector<bool>> visit;
    void dfs(vector<vector<int>>& image, int sr, int sc, int oldColor, int& newColor) {
        image[sr][sc] = newColor;
        visit[sr][sc] = true;
        for (int i = 0; i < 4; i++) {
            int tx = sr + arr[i], ty = sc + arr[i+1];
            if (tx >= 0 && tx < n && ty >= 0 && ty < m && visit[tx][ty] == false && image[tx][ty] == oldColor)
                dfs(image, tx, ty, oldColor, newColor);
```

```
        }
    }
};
```

739. Daily Temperatures [Medium]

Given a list of daily temperatures, produce a list that, for each day in the input, tells you how many days you would have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead.

For example, given the list temperatures = [73, 74, 75, 71, 69, 72, 76, 73], your output should be [1, 1, 4, 2, 1, 1, 0, 0].

Note: The length of temperatures will be in the range [1, 30000]. Each temperature will be an integer in the range [30, 100].

分析：用栈解决，i从0～len-1，每次将栈顶元素小于temperatures[i]的出栈，因为对于出栈的元素来说它已经找到了第一个大于它的值，剩余在栈中的都是未找到大于它本身的值的元素，则继续等待下一个temperatures[i]。每次将temperatures[i]压入栈中，等待接下来遇到比它大的值时出栈～将i与栈顶元素下标的差值保存在栈顶元素的下标所对应的ans中，最后返回ans即可～

```
class Solution {
public:
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        stack<pair<int, int>> s;
        int len = temperatures.size();
        vector<int> ans(len);
        for (int i = 0; i < len; i++) {
            while(!s.empty() && temperatures[i] > s.top().first) {
                ans[s.top().second] = i - s.top().second;
                s.pop();
            }
            s.push(pair<int, int>(temperatures[i], i));
        }
        return ans;
    }
};
```

740. Delete and Earn [Medium]

Given an array nums of integers, you can perform operations on the array.

In each operation, you pick any nums[i] and delete it to earn nums[i] points. After, you must delete every element equal to nums[i] – 1 or nums[i] + 1.

You start with 0 points. Return the maximum number of points you can earn by applying such operations.

Example 1: Input: nums = [3, 4, 2] Output: 6 Explanation: Delete 4 to earn 4 points, consequently 3 is also deleted. Then, delete 2 to earn 2 points. 6 total points are earned. Example 2: Input: nums = [2, 2, 3, 3, 3, 4] Output: 9 Explanation: Delete 3 to earn 3 points, deleting both 2 and the 4. Then, delete 3 again to earn 3 points, and 3 again to earn 3 points. 9 total points are earned. Note:

The length of nums is at most 20000. Each element nums[i] is an integer in the range [1, 10000].

题目大意：给一个数组，你可以拿起nums[i]并且删除nums[i]并得到nums[i]分数，但是你必须删除数组中所有的nums[i]-1和nums[i]+1，就是说nums[i]-1和nums[i]+1的分数是不可能得到了～求问你能够获得最大的分数会是多少～

分析：数组不是有序的，且在1～10000之间，所以先遍历数组，将i数字所拥有的个数保存在cnt[i]中。设立dp数组，含有10001个元素，（dp[i]表示遍历到i这个数的时候当前情况下的最大值。最后返回dp[10000]的值就是所求），dp[0] = 0, dp[1] = cnt[1]。i从2遍历到10000。对于dp[i]，因为如果要了i这个数就不能要i-1和i+1，所以当前i有两个选择：一，要i这个数带来的分数cnt[i] * i，那就不能要dp[i-1]只能要dp[i-2]。二，不要i带来的分数要dp[i-1]的分数。这两个选择取最大值，所以dp[i] = max(dp[i-1], cnt[i] * i + dp[i-2])，最后返回dp[10000]～

```
class Solution {
public:
    int deleteAndEarn(vector<int>& nums) {
        int n = nums.size();
        if (n == 0) return 0;
        vector<int> cnt(10001, 0), dp(10001, 0);
        for (int i = 0; i < n; i++) cnt[nums[i]]++;
        dp[1] = cnt[1];
        for (int i = 2; i <= 10000; i++)
            dp[i] = max(dp[i-1], cnt[i] * i + dp[i-2]);
        return dp[10000];
    }
};
```

744. Find Smallest Letter Greater Than Target [Easy]

Given a list of sorted characters letters containing only lowercase letters, and given a target letter target, find the smallest element in the list that is larger than the given target.

Letters also wrap around. For example, if the target is target = ‘z’ and letters = [‘a’, ‘b’], the answer is ‘a’.

Examples: Input: letters = [“c”, “f”, “j”] target = “a” Output: “c”

Input: letters = [“c”, “f”, “j”] target = “c” Output: “f”

Input: letters = [“c”, “f”, “j”] target = “d” Output: “f”

Input: letters = [“c”, “f”, “j”] target = “g” Output: “j”

Input: letters = [“c”, “f”, “j”] target = “j” Output: “c”

Input: letters = [“c”, “f”, “j”] target = “k” Output: “c” Note: letters has a length in range [2, 10000]. letters consists of lowercase letters, and contains at least 2 unique letters. target is a lowercase letter.

分析：用upper_bound返回第一个大于target的元素所在位置，如果这个位置等于letters.end()说明不存在，则返回letters的第一个值，否则返回it所在位置的元素值即可～

```
class Solution {
public:
    char nextGreatestLetter(vector<char>& letters, char target) {
        auto it = upper_bound(letters.begin(), letters.end(), target);
        return it == letters.end() ? letters[0] : *it;
    }
};
```

746. Min Cost Climbing Stairs [Easy]

On a staircase, the i-th step has some non-negative cost cost[i] assigned (0 indexed).

Once you pay the cost, you can either climb one or two steps. You need to find minimum cost to reach the top of the floor, and you can either start from the step with index 0, or the step with index 1.

Example 1: Input: cost = [10, 15, 20] Output: 15 Explanation: Cheapest is start on cost[1], pay that cost and go to the top. Example 2: Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1] Output: 6 Explanation: Cheapest is start on cost[0], and only step on 1s, skipping cost[3]. Note: cost will have a length in the range [2, 1000]. Every cost[i] will be an integer in the range [0, 999].

题目大意：爬n阶的楼梯，每层都有一个cost值（0～n-1），每次可以爬1层或者2层，求爬完全程的最小花费cost（可以从第0层开始也可以从第1层开始）

分析：dp数组，每次dp[i] = cost[i] + min(dp[i-1], dp[i-2])，最终返回dp[n-1]和dp[n-2]中较小的那个～

```
class Solution {
public:
    int minCostClimbingStairs(vector<int>& cost) {
        int n = cost.size();
        vector<int> dp(n);
        dp[0] = cost[0], dp[1] = cost[1];
        for (int i = 2; i < n; i++)
            dp[i] = cost[i] + min(dp[i-1], dp[i-2]);
        return min(dp[n-1], dp[n-2]);
    }
};
```

747. Largest Number At Least Twice of Others [Easy]

In a given integer array nums, there is always exactly one largest element.

Find whether the largest element in the array is at least twice as much as every other number in the array.

If it is, return the index of the largest element, otherwise return -1.

Example 1: Input: nums = [3, 6, 1, 0] Output: 1 Explanation: 6 is the largest integer, and for every other number in the array x, 6 is more than twice as big as x. The index of value 6 is 1, so we return 1. Example 2: Input: nums = [1, 2, 3, 4] Output: -1 Explanation: 4 isn't at least as big as twice the value of 3, so we return -1. Note: nums will have a length in the range [1, 50]. Every nums[i] will be an integer in the range [0, 99].

分析：找到最大值maxn、它对应的下标idx和次大值sec，如果次大值sec的两倍比maxn大说明不满足条件返回-1，否则返回idx

```
class Solution {
public:
    int dominantIndex(vector<int>& nums) {
        int maxn = INT_MIN, idx = -1, sec = INT_MIN;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] > maxn) {
                sec = maxn;
                maxn = nums[i];
                idx = i;
            } else if(nums[i] > sec){
                sec = nums[i];
            }
        }
        return sec * 2 > maxn ? -1 : idx;
    }
};
```

760. Find Anagram Mappings [Easy]

Given two lists Aand B, and B is an anagram of A. B is an anagram of A means B is made by randomizing the order of the elements in A.

We want to find an index mapping P, from A to B. A mapping P[i] = j means the ith element in A appears in B at index j.

These lists A and B may contain duplicates. If there are multiple answers, output any of them.

For example, given

A = [12, 28, 46, 32, 50] B = [50, 12, 32, 46, 28] We should return [1, 4, 3, 2, 0] as P[0] = 1 because the 0th element of A appears at B[1], and P[1] = 4 because the 1st element of A appears at B[4], and so on. Note:

A, B have equal lengths in range [1, 100]. A[i], B[i] are integers in range [0, 10^5].

题目大意：给两个数组A和B，B是A的同字母异序词，返回一个等长数组P，其中P[i] = j， 表示A的第i个元素在B的第j个元素处，如果有多个答案，返回一个即可～

分析：设置一个二维数组v，将数字i对应的在B数组中的下标放在v[i]中，这样遍历A数组，每次取出一个v[A[i]]放入ans数组对应的i下标处即可～

```
class Solution {
public:
    vector<int> anagramMappings(vector<int>& A, vector<int>& B) {
        int len = A.size();
        vector<int> ans(len), hash(len), v[100001];
        for (int i = 0; i < len; i++) v[B[i]].push_back(i);
        for (int i = 0; i < len; i++) {
            ans[i] = v[A[i]].back();
            v[A[i]].pop_back();
        }
        return ans;
    }
};
```

762. Prime Number of Set Bits in Binary Representation [Easy]

Given two integers L and R, find the count of numbers in the range [L, R] (inclusive) having a prime number of set bits in their binary representation. (Recall that the number of set bits an integer has is the number of 1s present when written in binary. For example, 21 written in binary is 10101 which has 3 set bits. Also, 1 is not a prime.) Example 1: Input: L = 6, R = 10 Output: 4 Explanation: 6 -> 110 (2 set bits, 2 is prime) 7 -> 111 (3 set bits, 3 is prime) 9 -> 1001 (2 set bits , 2 is prime) 10->1010 (2 set bits , 2 is prime) Example 2: Input: L = 10, R = 15 Output: 5 Explanation: 10 -> 1010 (2 set bits, 2 is prime) 11 -> 1011 (3 set bits, 3 is prime) 12 -> 1100 (2 set bits, 2 is prime) 13 -> 1101 (3 set bits, 3 is prime) 14 -> 1110 (3 set bits, 3 is prime) 15 -> 1111 (4 set bits, 4 is not prime) Note: L, R will be integers L <= R in the range [1, 10^6]. R – L will be at most 10000.

题目大意：给两个数L和R，在[L, R]区间寻找数字的二进制中1的个数是素数的数字个数。

分析：R不超过10的6次方，也就是不超过2的20次方，那么判断是否为素数只需判断是否等于2 3 5 7 11 13 17 19即可，用bitset也只需20位表示即可，用count函数计算1的个数，然后用cnt统计后返回～

```
class Solution {
public:
    int countPrimeSetBits(int L, int R) {
        int cnt = 0, hash[20] = {0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1};
        for (int i = L; i <= R; i++) {
            bitset<20> b(i);
            if(hash[b.count()]) cnt++;
        }
        return cnt;
    }
};
```

763. Partition Labels [Medium]

A string S of lowercase letters is given. We want to partition this string into as many parts as possible so that each letter appears in at most one part, and return a list of integers representing the size of these parts.

Example 1: Input: S = “ababcbacadefegdehijhklij” Output: [9,7,8] Explanation: The partition is “ababcbaca”, “defegde”, “hijhklij”. This is a partition so that each letter appears in at most one part. A partition like “ababcbacadefegde”, “hijhklij” is incorrect, because it splits S into less parts. Note:

S will have length in range [1, 500]. S will consist of lowercase letters (‘a’ to ‘z’) only.

题目大意：给出一个小写字母的字符串S. 我们想把这个字符串分成尽可能多的部分，这样每个字母最多只出现一个部分，并返回一个表示这些部分大小的整数列表。

分析：遍历字符串，找到S[i]在S中最后一次出现的位置标记为end，end位置是当前要切割的字串部分最短的结尾处，当i == end时候说明start～end可以组成一个最短部分串，将这个部分串的长度(end – start + 1)放入ans数组中，然后将start标记为下一个部分串的开始(end+1)，最后返回ans数组～

```
class Solution {
public:
    vector<int> partitionLabels(string S) {
        vector<int> ans;
        for (int i = 0, start = 0, end = 0; i < S.length(); i++) {
            end = max(end, (int)S.find_last_of(S[i]));
            if (i == end) {
                ans.push_back(end - start + 1);
                start = end + 1;
            }
        }
        return ans;
    }
};
```

766. Toeplitz Matrix [Easy]

A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same element.

Now given an M x N matrix, return True if and only if the matrix is Toeplitz.

Example 1:

Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]] Output: True Explanation: 1234 5123 9512

In the above grid, the diagonals are "[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]", and in each diagonal all elements are the same, so the answer is True. Example 2:

Input: matrix = [[1,2],[2,2]] Output: False Explanation: The diagonal "[1, 2]" has different elements. Note:

matrix will be a 2D array of integers. matrix will have a number of rows and columns in range [1, 20]. matrix[i][j] will be integers in range [0, 99].

题目大意：如果从左上角到右下角的每个对角线具有相同的元素，则矩阵是Toeplitz。给一个矩阵判断它是否是Toeplitz矩阵~

分析：对于每一个元素matrix[i][j]，都判断它是否等于它的右上角元素matrix[i-1][j-1]，如果不相等就返回false，否则循环结束后返回true

```
class Solution {
public:
    bool isToeplitzMatrix(vector<vector<int>>& matrix) {
        for (int i = 1; i < matrix.size(); i++)
            for (int j = 1; j < matrix[0].size(); j++)
                if (matrix[i][j] != matrix[i-1][j-1]) return false;
        return true;
    }
};
```

768. Max Chunks To Make Sorted II [Hard]

This question is the same as "Max Chunks to Make Sorted" except the integers of the given array are not necessarily distinct, the input array could be up to length 2000, and the elements could be up to 10**8.

Given an array arr of integers (not necessarily distinct), we split the array into some number of "chunks" (partitions), and individually sort each chunk. After concatenating them, the result equals the sorted array.

What is the most number of chunks we could have made?

Example 1:

Input: arr = [5,4,3,2,1] Output: 1 Explanation: Splitting into two or more chunks will not return the required result. For example, splitting into [5, 4], [3, 2, 1] will result in [4, 5, 1, 2, 3], which isn't sorted. Example 2:

Input: arr = [2,1,3,4,4] Output: 4 Explanation: We can split into two chunks, such as [2, 1], [3, 4, 4]. However, splitting into [2, 1], [3], [4], [4] is the highest number of chunks possible. Note:

arr will have length in range [1, 2000]. arr[i] will be an integer in range [0, 10**8].

题目大意：给一个排列，可能有重复元素，我们将数组拆分成一些“块”（分区），并对每个块进行单独排序。连接它们之后，结果等于排序后的数组。问最多能够分成多少个分区（块）

分析：因为有重复元素，可以考虑判断累加和的方式，排序后的数组前i个元素累加的和等于原数组前i个数累加的和时可以分为一个块～

```
class Solution {
public:
    int maxChunksToSorted(vector<int>& arr) {
        int sum1 = 0, sum2 = 0, ans = 0;
        vector<int> t = arr;
        sort(t.begin(), t.end());
        for(int i = 0; i < arr.size(); i++) {
            sum1 += t[i];
            sum2 += arr[i];
            if(sum1 == sum2) ans++;
        }
        return ans;
    }
};
```

769. Max Chunks To Make Sorted [Medium]

Given an array arr that is a permutation of [0, 1, ..., arr.length – 1], we split the array into some number of “chunks” (partitions), and individually sort each chunk. After concatenating them, the result equals the sorted array.

What is the most number of chunks we could have made?

Example 1:

Input: arr = [4,3,2,1,0] Output: 1 Explanation: Splitting into two or more chunks will not return the required result. For example, splitting into [4, 3], [2, 1, 0] will result in [3, 4, 0, 1, 2], which isn’t sorted. Example 2:

Input: arr = [1,0,2,3,4] Output: 4 Explanation: We can split into two chunks, such as [1, 0], [2, 3, 4]. However, splitting into [1, 0], [2], [3], [4] is the highest number of chunks possible. Note:

arr will have length in range [1, 10]. arr[i] will be a permutation of [0, 1, ..., arr.length – 1].

题目大意：给0～arr.length-1的一个排列，我们将数组拆分成一些“块”（分区），并对每个块进行单独排序。 连接它们之后，结果等于排序后的数组。问最多能够分成多少个分区（块）

分析：因为数组的排序后正确顺序应该是arr[i]处的数是i。所以，遍历数组，每次将最大的那个值标记为maxn，maxn必须在i处才能满足对0～i数字排序后能够恰好是正确的位置，此时ans+1，表示前面的可以组为一个“块”，最后ans即为所求的值～ 再解释详细些：maxn是第0～i个数字中的最大值，遍历的过程中如果maxn==i，就保证了前面i-1个数字必然都比maxn小（因为maxn是0～i中的最大值），则第0～i个数字必然能排列成正确顺序，以此类推，找到下一个满足maxn==i的地方（记为j），则i+1～j又能分为一个块...直到遍历到最后一个数为止得到答案～

```
class Solution {
public:
    int maxChunksToSorted(vector<int>& arr) {
        int ans = 0;
        for (int i = 0, maxn = 0; i < arr.size(); i++) {
            maxn = max(arr[i], maxn);
            if (maxn == i) ans++;
        }
        return ans;
    }
};
```

771. Jewels and Stones [Easy]

You are given strings J representing the types of stones that are jewels, and S representing the stones you have. Each character in S is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in J are guaranteed distinct, and all characters in J and S are letters. Letters are case sensitive, so “a” is considered a different type of stone from “A”.

Example 1:

Input: J = “aA”, S = “aAAbbbb” Output: 3 Example 2:

Input: J = “z”, S = “ZZ” Output: 0 Note:

S and J will consist of letters and have length at most 50. The characters in J are distinct.

题目大意：J是珠宝，S是自己拥有的石头，判断自己拥有的石头中有多少个珠宝～ 分析：将J中的所有字符在hash数组中标记为1，再遍历S，统计hash == 1的个数即为所求～

```cpp
class Solution {
public:
    int numJewelsInStones(string J, string S) {
        int ans = 0, hash[256] = {0};
        for (auto i : J) hash[i] = 1;
        for (auto i : S) if (hash[i]) ans++;
        return ans;
    }
};
```

775. Global and Local Inversions [Medium]

We have some permutation A of [0, 1, ..., N – 1], where N is the length of A.

The number of (global) inversions is the number of i < j with 0 <= i < j < N and A[i] > A[j].

The number of local inversions is the number of i with 0 <= i < N and A[i] > A[i+1].

Return true if and only if the number of global inversions is equal to the number of local inversions.

Example 1:

Input: A = [1,0,2] Output: true Explanation: There is 1 global inversion, and 1 local inversion. Example 2:

Input: A = [1,2,0] Output: false Explanation: There are 2 global inversions, and 1 local inversion. Note:

A will be a permutation of [0, 1, ..., A.length – 1]. A will have length in range [1, 5000]. The time limit for this problem has been reduced.

题目大意：全局反转次数是0 <= i <j <N且A [i]> A [j]的i <j的个数，局部反转的次数是0 <= i <N且A [i]> A [i + 1]的i的个数。当且仅当全局反转的次数等于本地反转的次数时才返回true。

分析：局部反转属于全局反转，所以说这个数组里只允许出现相邻的两个数字是大小反转的，其他的若是间隔的两个数字，都只能是从小到大的顺序~

A[i]必定只能在它本应在的位置或这个位置的左边一个或者右边一个，不然必定会把比它大的两个数字挤到前面或者把比它小的两个数字挤到后面，因为题目中已经说明它所有的数字是0～n-1的一个排列，那么A[i]-i的绝对值必定<=1，否则就不满足条件～

```cpp
class Solution {
public:
    bool isIdealPermutation(vector<int>& A) {
        for (int i = 0; i < A.size(); i++)
            if (abs(A[i] - i) >= 2) return false;
        return true;
    }
};
```

779. K-th Symbol in Grammar [Medium]

On the first row, we write a 0. Now in every subsequent row, we look at the previous row and replace each occurrence of 0 with 01, and each occurrence of 1 with 10.

Given row N and index K, return the K-th indexed symbol in row N. (The values of K are 1-indexed.) (1 indexed).

Examples: Input: N = 1, K = 1 Output: 0

Input: N = 2, K = 1 Output: 0

Input: N = 2, K = 2 Output: 1

Input: N = 4, K = 5 Output: 1

Explanation: row 1: 0 row 2: 01 row 3: 0110 row 4: 01101001 Note:

N will be an integer in the range [1, 30]. K will be an integer in the range [1, 2^(N-1)].

题目大意：在第一行，我们写一个0.现在在后面的每一行中，我们看前一行，用01代替0出现的每一个，每一次出现1代表10。现在问第N行第K数字是什么数字

分析：用递归，已知N == 1的时候返回0，为了知道第N行第K个数字的值，只要知道它在第N-1行的第(K+1)/2个数字对应的值即可，因为0对应01，1对应10，那么如果K是奇数只需和原数字相同即可，如果K是偶数只需对原对应数字取反即可～

```
class Solution {
public:
    int kthGrammar(int N, int K) {
        if (N == 1) return 0;
        return (K % 2 == 0) ? !kthGrammar(N-1, (K+1)/2) : kthGrammar(N-1, (K+1)/2);
    }
};
```