TOP100

**1.两数之和**

给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那 两个 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

示例:

给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9

所以返回 [0, 1]

**解答：使用hashmap映射，若**

class Solution {

    public int[] twoSum(int[] nums, int target) {

        int[] res=new int[2];

        HashMap<Integer,Integer> hm=new HashMap<>();

        for(int i=0;i<nums.length;i++){

            int dif=target-nums[i];

            if(hm.get(dif)!=null){

                res[0]=i;

                res[1]=hm.get(dif);

                break;

            }

            hm.put(nums[i],i);

        }

        return res;

    }

}

**2.两数相加**

给出两个 非空 的链表用来表示两个非负的整数。其中，它们各自的位数是按照 逆序 的方式存储的，并且它们的每个节点只能存储 一位 数字。

如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。

您可以假设除了数字 0 之外，这两个数都不会以 0 开头。

示例：

输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)

输出：7 -> 0 -> 8

原因：342 + 465 = 807

要点：注意两者长度不同，和在最后一位需要进位的情况

class Solution {

    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

        int tag=0;

        int t1=l1.val,t2=l2.val;

        ListNode res=new ListNode((l1.val+l2.val)%10);//get the first number

        ListNode head=res;

        tag=(l1.val+l2.val)/10;

        l1=l1.next;

        l2=l2.next;

        while(l1!=null||l2!=null){

            t1=0;t2=0;

            if(l1!=null){

                t1=l1.val;

                l1=l1.next;

            }

            if(l2!=null){

                t2=l2.val;

                l2=l2.next;

            }

            ListNode tempnode=new ListNode((t1+t2+tag)%10);

            res.next=tempnode;

            res=res.next;

            tag=(t1+t2+tag)/10;

        }

        if((t1+t2+tag)>=10){//判断最后一位是否需要进位（例如999+1）

            ListNode tempnode=new ListNode(1);

            res.next=tempnode;

        }

        return head;

    }

}

**3.无重复字符的最长子串**

给定一个字符串，请你找出其中不含有重复字符的 最长子串 的长度。

示例 1:

输入: "abcabcbb"

输出: 3

解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

示例 2:

输入: "bbbbb"

输出: 1

解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。

示例 3:

输入: "pwwkew"

输出: 3

解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。

  请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

要点：**滑动窗口配合HashMap**

class Solution {

    public int lengthOfLongestSubstring(String s) {

        if (s.length()==0) return 0;

        HashMap<Character, Integer> map = new HashMap<Character, Integer>();

        int max = 0;

        int left = 0;

        for(int i = 0; i < s.length(); i ++){

            if(map.containsKey(s.charAt(i))){

                left = Math.max(left,map.get(s.charAt(i)) + 1);

            }

            map.put(s.charAt(i),i);

            max = Math.max(max,i-left+1);

        }

        return max;

    }

}

时间复杂度： *O*(*N*)，

空间复杂度： *O*(∣Σ∣)

**4.寻找两个正序数组的中位数**

给定两个大小为 m 和 n 的正序（从小到大）数组 nums1 和 nums2。请你找出并返回这两个正序数组的中位数。

进阶：你能设计一个时间复杂度为 O(log (m+n)) 的算法解决此问题吗？

示例 1：

输入：nums1 = [1,3], nums2 = [2]

输出：2.00000

解释：合并数组 = [1,2,3] ，中位数 2

示例 2：

输入：nums1 = [1,2], nums2 = [3,4]

输出：2.50000

解释：合并数组 = [1,2,3,4] ，中位数 (2 + 3) / 2 = 2.5

示例 3：

输入：nums1 = [0,0], nums2 = [0,0]

输出：0.00000

示例 4：

输入：nums1 = [], nums2 = [1]

输出：1.00000

示例 5：

输入：nums1 = [2], nums2 = []

输出：2.00000

**要点：解题不难，难在优化到时间复杂度为log(m+n)，这里只给了m+n的。**

class Solution {

public double findMedianSortedArrays(int[] A, int[] B) {

    int m = A.length;

    int n = B.length;

    int len = m + n;

    int left = -1, right = -1;

    int aStart = 0, bStart = 0;

    for (int i = 0; i <= len / 2; i++) {

        left = right;\*\*\*\*\*\*\*\*\*\*\*\*

        if (aStart < m && (bStart >= n || A[aStart] < B[bStart])) {

            right = A[aStart++];

        } else {

            right = B[bStart++];

        }

    }

    if ((len & 1) == 0)

        return (left + right) / 2.0;

    else

        return right;

}

}

**考虑奇偶两者情况，对于奇：中位数应该在**len/2**处，对于偶：中位数应该在**len/2-1**和**len/2**处。对于两种，均应该遍历在**len/2**次。那么设置该长度的循环，找到两组数组中该位置的数。取right作为更新的位置，对于两个数组，遍历时所发生的情况很多，但是结果只有两种，要么是数组A的数，要么是数组B的数，那么考虑情况为数组A锁可能发生的情况，剩下的则是数组B的数。对于数组A,首先一定是处在A的长度范围内，所以取一个并，对于数组B，有可能在长度内，有可能在长度外，如果在长度内，条件就是**A[aStart] < B[bStart]，**如果超出数组B，则肯定是A数组的数，因此条件汇总成**(aStart < m && (bStart >= n || A[aStart] < B[bStart]))。剩下的是else 为B。

**5.最长回文子串**

给定一个字符串 s，找到 s 中最长的回文子串。你可以假设 s 的最大长度为 1000。

示例 1：

输入: "babad"

输出: "bab"

注意: "aba" 也是一个有效答案。

示例 2：

输入: "cbbd"

输出: "bb"

要点：中间开花和动态规划

class Solution {

    public String longestPalindrome(String s) {

        int max=0;

        String res="";

        if(s.length()<=1) return s;

        for(int i=0;i<s.length();i++){

            String temp=judge(s,i);

            if(temp.length()>max){

                max=temp.length();

                res=temp;

            }

        }

        return res;

    }

    String judge(String s,int i){

        int left=i-1,right=i+1;

        //中间对称

        while(left>=0&&right<s.length()){

            if(s.charAt(left)!=s.charAt(right)){

                break;

            }

            left--;

            right++;

        }

        String s1=s.substring(left+1,right);

        left=i;

        right=i+1;

        //两边对称

        while(left>=0&&right<s.length()){

            if(s.charAt(left)!=s.charAt(right)){

                break;

            }

            left--;

            right++;

        }

        String s2=s.substring(left+1,right);

        return (s1.length()>s2.length())?s1:s2;

    }

}

动态规划

public class Solution {

public String longestPalindrome(String s) {

// 特判

int len = s.length();

if (len < 2) {

return s;

}

int maxLen = 1;

int begin = 0;

// dp[i][j] 表示 s[i, j] 是否是回文串

boolean[][] dp = new boolean[len][len];

char[] charArray = s.toCharArray();

for (int i = 0; i < len; i++) {

dp[i][i] = true;

}

for (int j = 1; j < len; j++) {

for (int i = 0; i < j; i++) {

if (charArray[i] != charArray[j]) {

dp[i][j] = false;

} else {

if (j - i < 3) {

dp[i][j] = true;

} else {

dp[i][j] = dp[i + 1][j - 1];

}

}

// 只要 dp[i][j] == true 成立，就表示子串 s[i..j] 是回文，此时记录回文长度和起始位置

if (dp[i][j] && j - i + 1 > maxLen) {

maxLen = j - i + 1;

begin = i;

}

}

}

return s.substring(begin, begin + maxLen);

}

}

**6.整数反转**

要点：整数溢出处理

给出一个 32 位的有符号整数，你需要将这个整数中每位上的数字进行反转。

假设我们的环境只能存储得下 32 位的有符号整数，则其数值范围为 [−231,  231− 1]。请根据这个假设，如果反转后整数溢出那么就返回 0。

class Solution {

    public int reverse(int x) {

        int res=0;

        while(x!=0){

            int pop=x%10;

            x=x/10;

            if(res>Integer.MAX\_VALUE/10||(res==Integer.MAX\_VALUE/10&&pop>7)) return 0;

            if(res<Integer.MIN\_VALUE/10||(res==Integer.MIN\_VALUE/10&&pop<-8)) return 0;

            res=res\*10+pop;

        }

        return res;

    }

}

时间复杂度：O(log(x))

空间复杂度：O(1)O(1)。

**7.字符串转换整数（atoi）**

请你来实现一个 atoi 函数，使其能将字符串转换成整数。

首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。接下来的转化规则如下：

如果第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字字符组合起来，形成一个有符号整数。

假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成一个整数。

该字符串在有效的整数部分之后也可能会存在多余的字符，那么这些字符可以被忽略，它们对函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换，即无法进行有效转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0 。

提示：

本题中的空白字符只包括空格字符 ' ' 。

假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 [−231,  231 − 1]。如果数值超过这个范围，请返回  INT\_MAX (231 − 1) 或 INT\_MIN (−231)

要点：先找到第一个不为空格的位置，判断其符号作为sign，然后使用while判断连续的数字，若不是，则会跳出。因为要检查超出整型范围，因此使用更大范围的long，最后再从大向小进行强转。

class Solution {

    public int myAtoi(String s) {

        //if(s.length()==0) return 0;

        int i=0,n=s.length();

        while(i<n&&s.charAt(i)==' '){

            i++;

        }

         int sign=1;

         long res=0;

        if(i<n&&(s.charAt(i)=='-'||s.charAt(i)=='+')){

            sign=(s.charAt(i++)=='-')?(-1):1;

        }

        while(i<n&&s.charAt(i)>='0'&&s.charAt(i)<='9'){

            res=res\*10+(s.charAt(i++)-'0');

            if(res\*sign>Integer.MAX\_VALUE) return Integer.MAX\_VALUE;

            if(res\*sign<Integer.MIN\_VALUE) return Integer.MIN\_VALUE;

        }

        return (int)res\*sign;

    }

}

**8.正则表达式匹配**

给你一个字符串 s 和一个字符规律 p，请你来实现一个支持 '.' 和 '\*' 的正则表达式匹配。

'.' 匹配任意单个字符

'\*' 匹配零个或多个前面的那一个元素

所谓匹配，是要涵盖 整个 字符串 s的，而不是部分字符串。

示例 1：

输入：s = "aa" p = "a"

输出：false

解释："a" 无法匹配 "aa" 整个字符串。

示例 2:

输入：s = "aa" p = "a\*"

输出：true

解释：因为 '\*' 代表可以匹配零个或多个前面的那一个元素, 在这里前面的元素就是 'a'。因此，字符串 "aa" 可被视为 'a' 重复了一次。

示例 3：

输入：s = "ab" p = ".\*"

输出：true

解释：".\*" 表示可匹配零个或多个（'\*'）任意字符（'.'）。

示例 4：

输入：s = "aab" p = "c\*a\*b"

输出：true

解释：因为 '\*' 表示零个或多个，这里 'c' 为 0 个, 'a' 被重复一次。因此可以匹配字符串 "aab"。

示例 5：

输入：s = "mississippi" p = "mis\*is\*p\*."

输出：false

要点：

**9.盛最多水的容器**

给你 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0) 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

说明：你不能倾斜容器。

输入：[1,8,6,2,5,4,8,3,7]

输出：49

解释：图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

要点：双指针，左右为边界，谁小谁递进变化

class Solution {

    public int maxArea(int[] height) {

        int left=0,right=height.length-1;

        int max=0;

        while(left<right){

            int capacity=((height[left]<height[right])?height[left]:height[right])\*(right-left);

            if(capacity>max)

                max=capacity;

            if(height[left]>=height[right])

                --right;

            else

                ++left;

        }

        return max;

    }

}

**10.罗马数字转整数**

例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。 27 写做  XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。

X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。

C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内

输入: "LVIII"

输出: 58

解释: L = 50, V= 5, III = 3.

要点：如果钱一个数字小于后一个数字，证明要减，否则一定是加。例如IV表示4，可以理解为-1+5，循环遍历后，要再加一次，因为每次迭代的是前一次的数字结果。

class Solution {

    public int romanToInt(String s) {

        int sum = 0;

        int preNum = getValue(s.charAt(0));

        for(int i = 1;i < s.length(); i ++) {

            int num = getValue(s.charAt(i));

            if(preNum < num) {

                sum -= preNum;

            } else {

                sum += preNum;

            }

            preNum = num;

        }

        sum += preNum;

        return sum;

    }

    private int getValue(char ch) {

        switch(ch) {

            case 'I': return 1;

            case 'V': return 5;

            case 'X': return 10;

            case 'L': return 50;

            case 'C': return 100;

            case 'D': return 500;

            case 'M': return 1000;

            default: return 0;

        }

    }

}

**11.最长公共前缀**

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 ""。

示例 1:

输入: ["flower","flow","flight"]

输出: "fl"

示例 2:

输入: ["dog","racecar","car"]

输出: ""

解释: 输入不存在公共前缀。

class Solution {

public String longestCommonPrefix(String[] strs) {

if(strs.length<1) return "";

if(strs.length==1) return strs[0];

StringBuilder sb=new StringBuilder();

for(int j=0;j<strs[0].length();j++){

char tempchar=strs[0].charAt(j);

for(int i=0;i<strs.length;i++){

if(j>=strs[i].length()||strs[i].charAt(j)!=tempchar)

return sb.toString();

}

sb.append(tempchar);

}

return sb.toString();

}

}

**12.三数之和**

给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有满足条件且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例：

给定数组 nums = [-1, 0, 1, 2, -1, -4]，

满足要求的三元组集合为：

[

[-1, 0, 1],

[-1, -1, 2]

]

**要点：处理重复问题**

class Solution {

    List<List<Integer>> res=new LinkedList<>();

    public List<List<Integer>> threeSum(int[] nums) {

        int n=nums.length;

        Arrays.sort(nums);

        // 枚举 a

        for (int first = 0; first < n; ++first) {

            // 需要和上一次枚举的数不相同

            if (first > 0 && nums[first] == nums[first - 1]) {

                continue;

            }

            // c 对应的指针初始指向数组的最右端

            int third = n - 1;

            int target = -nums[first];

            // 枚举 b

            for (int second = first + 1; second < n; ++second) {

                // 需要和上一次枚举的数不相同

                if (second > first + 1 && nums[second] == nums[second - 1]) {

                    continue;

                }

                // 需要保证 b 的指针在 c 的指针的左侧

                while (second < third && nums[second] + nums[third] > target) {

                    --third;

                }

                // 如果指针重合，随着 b 后续的增加

                // 就不会有满足 a+b+c=0 并且 b<c 的 c 了，可以退出循环

                if (second == third) {

                    break;

                }

                if (nums[second] + nums[third] == target) {

                    List<Integer> list = new ArrayList<Integer>();

                    list.add(nums[first]);

                    list.add(nums[second]);

                    list.add(nums[third]);

                    res.add(list);

                }

            }

        }

        return res;

    }

}

**13.电话号码的字母组合**

示例:

输入："23"

输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

要点：队列 || 回溯

class Solution {

    public List<String> letterCombinations(String digits) {

        if(digits.length()==0)

            return new ArrayList<>();

        List<String> res = new ArrayList<>();

        //先往队列中加入一个空字符

        res.add("");

            for(int i=0;i<digits.length();i++) {

            //由当前遍历到的字符，取字典表中查找对应的字符串

            String letters = getletter(digits.charAt(i)-'0');

            int size = res.size();

            //计算出队列长度后，将队列中的每个元素挨个拿出来

            for(int j=0;j<size;j++) {

                //每次都从队列中拿出第一个元素

                String tmp = res.remove(0);

                //然后跟"def"这样的字符串拼接，并再次放到队列中

                for(int k=0;k<letters.length();k++) {

                    res.add(tmp+letters.charAt(k));

                }

            }

        }

        return res;

    }

    String getletter(int num){

        switch(num){

            case 2:

            return "abc";

            case 3:

            return "def";

            case 4:

            return "ghi";

            case 5:

            return "jkl";

            case 6:

            return "mno";

            case 7:

            return "pqrs";

            case 8:

            return "tuv";

            case 9:

            return "wxyz";

            default:

            return "";

        }

    }

}

**回溯**

class Solution {

    //一个映射表，第二个位置是"abc“,第三个位置是"def"。。。

    //这里也可以用map，用数组可以更节省点内存

    String[] letter\_map = {" ","\*","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};

    public List<String> letterCombinations(String digits) {

        //注意边界条件

        if(digits==null || digits.length()==0) {

            return new ArrayList<>();

        }

        iterStr(digits, new StringBuilder(), 0);

        return res;

    }

    //最终输出结果的list

    List<String> res = new ArrayList<>();

    //递归函数

    void iterStr(String str, StringBuilder letter, int index) {

        //递归的终止条件，注意这里的终止条件看上去跟动态演示图有些不同，主要是做了点优化

        //动态图中是每次截取字符串的一部分，"234"，变成"23"，再变成"3"，最后变成""，这样性能不佳

        //而用index记录每次遍历到字符串的位置，这样性能更好

        if(index == str.length()) {

            res.add(letter.toString());

            return;

        }

        //获取index位置的字符，假设输入的字符是"234"

        //第一次递归时index为0所以c=2，第二次index为1所以c=3，第三次c=4

        //subString每次都会生成新的字符串，而index则是取当前的一个字符，所以效率更高一点

        char c = str.charAt(index);

        //map\_string的下表是从0开始一直到9， c-'0'就可以取到相对的数组下标位置

        //比如c=2时候，2-'0'，获取下标为2,letter\_map[2]就是"abc"

        int pos = c - '0';

        String map\_string = letter\_map[pos];

        //遍历字符串，比如第一次得到的是2，页就是遍历"abc"

        for(int i=0;i<map\_string.length();i++) {

            //调用下一层递归，用文字很难描述，请配合动态图理解

            letter.append(map\_string.charAt(i));

            //如果是String类型做拼接效率会比较低

            //iterStr(str, letter+map\_string.charAt(i), index+1);

            iterStr(str, letter, index+1);

            letter.deleteCharAt(letter.length()-1);//删除当前添加的最后一位元素，进行下一位元素的添加

        }

    }

}

**14.删除链表的倒数第N个节点**

给定一个链表，删除链表的倒数第 n 个节点，并且返回链表的头结点。

示例：

给定一个链表: 1->2->3->4->5, 和 n = 2.

当删除了倒数第二个节点后，链表变为 1->2->3->5.

要点：使用栈，倒数第N个就是弹出栈顶几次,Deque

class Solution {

    public ListNode removeNthFromEnd(ListNode head, int n) {

        ListNode headhead=new ListNode(0,head);

        Deque<ListNode> stack=new LinkedList<ListNode>();

        ListNode cur=headhead;

        while(cur!=null){

            stack.push(cur);

            cur=cur.next;

        }

        for(int i=0;i<n;i++){

            stack.pop();

        }

        stack.peek().next=stack.peek().next.next;

        return headhead.next;

    }

}

一次遍历（双指针）

class Solution {

public ListNode removeNthFromEnd(ListNode head, int n) {

ListNode dummy = new ListNode(0, head);

ListNode first = head;

ListNode second = dummy;

for (int i = 0; i < n; ++i) {

first = first.next;

}

while (first != null) {

first = first.next;

second = second.next;

}

second.next = second.next.next;

ListNode ans = dummy.next;

return ans;

}

}

**15.有效括号**

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

注意空字符串可被认为是有效字符串。

要点：此题可扩展，例如（[]）为无效括号。

class Solution {

    public boolean isValid(String s) {

        if(s.isEmpty()) return true;

        Stack<Character> stack=new Stack<>();

        for(int i=0;i<s.length();i++){

            char temp=s.charAt(i);

            if(temp=='('){

                stack.add(')');

            }else if(temp=='['){

                stack.add(']');

            }else if(temp=='{'){

                stack.add('}');

            }else if(stack.isEmpty()||temp!=stack.pop()){

                return false;

            }

        }

        return stack.isEmpty();

    }

}

**16.合并两个有序链表**

将两个升序链表合并为一个新的 升序 链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

示例：

输入：1->2->4, 1->3->4

输出：1->1->2->3->4->4

要点：递归，迭代法类似于**寻找中位数**那题

class Solution {

    public ListNode mergeTwoLists(ListNode l1, ListNode l2)

    {

       if (l1 == null) {

            return l2;

        }

        else if (l2 == null) {

            return l1;

        }

        else if (l1.val < l2.val) {

            l1.next = mergeTwoLists(l1.next, l2);

            return l1;

        }

        else {

            l2.next = mergeTwoLists(l1, l2.next);

            return l2;

        }

    }

}

**17. 括号生成**

数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 有效的 括号组合。

示例：

输入：n = 3

输出：[

"((()))",

"(()())",

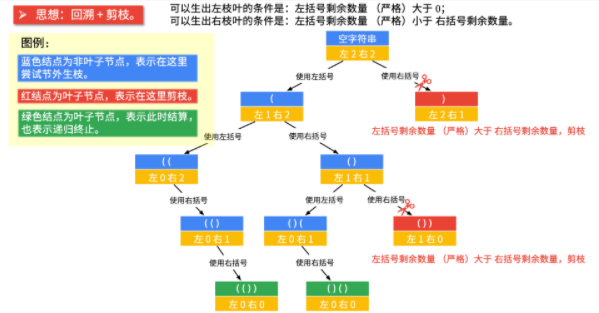
"(())()",

"()(())",

"()()()"

]

要点：采用树结构，回溯加剪枝。



class Solution {

    public List<String> generateParenthesis(int n) {

        List<String> res=new ArrayList<>();

        if(n==0)

            return res;

        dfs(res,n,n,"");

        return res;

    }

    void dfs(List<String> res,int left,int right,String ans){

        if(left==0&&right==0){

            res.add(ans);

            return;

        }

        if(right<left)

            return;

        if(left>0)

            dfs(res,left-1,right,ans+"(");

        if(right>0)

            dfs(res,left,right-1,ans+")");

    }

}

**18.合并K个升序链表**

**19.删除排序数组中的重复项**

给定一个排序数组，你需要在 原地 删除重复出现的元素，使得每个元素只出现一次，返回移除后数组的新长度。

不要使用额外的数组空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

示例 1:

给定数组 nums = [1,1,2],

函数应该返回新的长度 2, 并且原数组 nums 的前两个元素被修改为 1, 2。

你不需要考虑数组中超出新长度后面的元素。

要点：在原数组上覆盖，利用一个单指针指向索引即可。

class Solution {

    public int removeDuplicates(int[] nums) {

        if(nums.length<=1) return nums.length;

        int left=0;

        for(int i=1;i<nums.length;i++){

            if(nums[left]!=nums[i]){

                left++;

                nums[left]=nums[i];

            }

        }

        return left+1;

    }

}

**20.实现strStr()**

实现 strStr() 函数。

给定一个 haystack 字符串和一个 needle 字符串，在 haystack 字符串中找出 needle 字符串出现的第一个位置 (从0开始)。如果不存在，则返回  -1。

示例 1:

输入: haystack = "hello", needle = "ll"

输出: 2

示例 2:

输入: haystack = "aaaaa", needle = "bba"

输出: -1

说明:

当 needle 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当 needle 是空字符串时我们应当返回 0 。这与C语言的 strstr() 以及 Java的 indexOf() 定义相符。

**21.两数相除**

给定两个整数，被除数 dividend 和除数 divisor。将两数相除，要求不使用乘法、除法和 mod 运算符。

返回被除数 dividend 除以除数 divisor 得到的商。

整数除法的结果应当截去（truncate）其小数部分，例如：truncate(8.345) = 8 以及 truncate(-2.7335) = -2

示例 1:

输入: dividend = 10, divisor = 3

输出: 3

解释: 10/3 = truncate(3.33333..) = truncate(3) = 3

示例 2:

输入: dividend = 7, divisor = -3

输出: -2

解释: 7/-3 = truncate(-2.33333..) = -2

**22.搜索旋转排序数组\*\*\*\*\*\*\*\*\*\***

给你一个整数数组 nums ，和一个整数 target 。

该整数数组原本是按升序排列，但输入时在预先未知的某个点上进行了旋转。（例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] ）。

请你在数组中搜索 target ，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

示例 1：

输入：nums = [4,5,6,7,0,1,2], target = 0

输出：4

示例 2：

输入：nums = [4,5,6,7,0,1,2], target = 3

输出：-1

示例 3：

输入：nums = [1], target = 0

输出：-1

要点：

**23.在排序数组中查找元素的第一个和最后一个位置**

给定一个按照升序排列的整数数组 nums，和一个目标值 target。找出给定目标值在数组中的开始位置和结束位置。

你的算法时间复杂度必须是 O(log n) 级别。

如果数组中不存在目标值，返回 [-1, -1]。

示例 1:

输入: nums = [5,7,7,8,8,10], target = 8

输出: [3,4]

示例 2:

输入: nums = [5,7,7,8,8,10], target = 6

输出: [-1,-1]

**要点：二分法**

class Solution {

    public int[] searchRange(int[] nums, int target) {

        int left=0,right=nums.length-1;

        int[] res={-1,-1};

        while(left<=right){

            int mid=left+(right-left)/2;

            if(nums[mid]>target){

                right=mid-1;

            }else if(nums[mid]<target){

                left=mid+1;

            }else{

                res=getindex(nums,mid,target);

                break;

            }

        }

        return res;

    }

    int[] getindex(int[] nums, int mid,int target){

        int left=mid,right=mid;

        while(left>=0&&nums[left]==target){

            left--;

        }

        while(right<nums.length&&nums[right]==target){

            right++;

        }

        int[] res={left+1,right-1};

        return res;

    }

}

**24.将有序数组转换为二叉搜索树**

将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。

示例:

给定有序数组: [-10,-3,0,5,9],

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

0

/ \

-3 9

/ /

-10 5

要点：中序遍历

class Solution {

    public TreeNode sortedArrayToBST(int[] nums) {

        return dfs(nums, 0, nums.length - 1);

    }

    private TreeNode dfs(int[] nums, int lo, int hi) {

        if (lo > hi) {

            return null;

        }

        // 以升序数组的中间元素作为根节点 root。

        int mid = lo + (hi - lo) / 2;

        TreeNode root = new TreeNode(nums[mid]);

        // 递归的构建 root 的左子树与右子树。

        root.left = dfs(nums, lo, mid - 1);

        root.right = dfs(nums, mid + 1, hi);

        return root;

    }

}

**25.二叉树的最大深度**

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例：

给定二叉树 [3,9,20,null,null,15,7]，

3

/ \

9 20

/ \

15 7

返回它的最大深度 3 。

要点：迭代

class Solution {

    public int maxDepth(TreeNode root) {

        if(root==null)

            return 0;

        else{

            int l=maxDepth(root.left);

            int r=maxDepth(root.right);

            return Math.max(l,r)+1;

        }

    }

}

**26.删除链表中的节点**

请编写一个函数，使其可以删除某个链表中给定的（非末尾）节点。传入函数的唯一参数为 **要被删除的节点** 。

示例 1：

输入：head = [4,5,1,9], node = 5

输出：[4,1,9]

解释：给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.

示例 2：

输入：head = [4,5,1,9], node = 1

输出：[4,5,9]

解释：给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

要点：

class Solution {

    public void deleteNode(ListNode node) {

node.val=node.next.val;

node.next=node.next.next;    }

}

**27.反转字符串**

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 char[] 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

你可以假设数组中的所有字符都是 ASCII 码表中的可打印字符。

示例 1：

输入：["h","e","l","l","o"]

输出：["o","l","l","e","h"]

示例 2：

输入：["H","a","n","n","a","h"]

输出：["h","a","n","n","a","H"]

要点：双指针

class Solution {

    public void reverseString(char[] s) {

        int i=0,j=s.length-1;

        while(i<j){

            char temp=s[i];

            s[i]=s[j];

            s[j]=temp;

            i++;

            j--;

        }

    }

}

**28.Excel表序列号**

给定一个Excel表格中的列名称，返回其相应的列序号。

例如，

A -> 1

B -> 2

C -> 3

...

Z -> 26

AA -> 27

AB -> 28

...

示例 1:

输入: "A"

输出: 1

示例 2:

输入: "AB"

输出: 28

示例 3:

输入: "ZY"

输出: 701

要点：因为有 26 个字母，所以相当于 26 进制，每 26 个数则向前进一位，所以每遍历一位则ans = ans \* 26 + num。

class Solution {

    public int titleToNumber(String s) {

        int ans = 0;

        for(int i=0;i<s.length();i++) {

            int num = s.charAt(i) - 'A' + 1;

            ans = ans \* 26 + num;

        }

        return ans;

    }

}

**29.杨辉三角**

给定一个非负整数 numRows，生成杨辉三角的前 numRows 行。

**输入:** 5

**输出:**

[

[1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]

要点：利用关系构建

class Solution {

    public List<List<Integer>> generate(int numRows) {

        List<List<Integer>> res=new ArrayList<>();

        if(numRows==0) return res;

        res.add(List.of(1));

        if(numRows==1) return res;

        res.add(List.of(1,1));

        if(numRows==2) return res;

        for(int i=2;i<numRows;i++){

            List<Integer> temp =new ArrayList<>(i+1);

            temp.add(1);

            for(int j=1;j<i;j++){

                temp.add(res.get(i-1).get(j-1)+res.get(i-1).get(j));

            }

            temp.add(1);

            res.add(temp);

        }

        return res;

    }

}

**30.岛屿数量**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

示例 1：

输入：grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

输出：1

示例 2：

输入：grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

输出：3

提示：

m == grid.length

n == grid[i].length

1 <= m, n <= 300

grid[i][j] 的值为 '0' 或 '1'

要点：DFS

class Solution {

    public int numIslands(char[][] grid) {

        int count = 0;

        for(int i = 0; i < grid.length; i++) {

            for(int j = 0; j < grid[0].length; j++) {

                if(grid[i][j] == '1'){

                    dfs(grid, i, j);

                    count++;

                }

            }

        }

        return count;

    }

    private void dfs(char[][] grid, int i, int j){

        if(i < 0 || j < 0 || i >= grid.length || j >= grid[0].length || grid[i][j] == '0') return;

        grid[i][j] = '0';

        dfs(grid, i + 1, j);

        dfs(grid, i, j + 1);

        dfs(grid, i - 1, j);

        dfs(grid, i, j - 1);

    }

}

**31.x的平方根**

实现 int sqrt(int x) 函数。

计算并返回 x 的平方根，其中 x 是非负整数。

由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

示例 1:

输入: 4

输出: 2

示例 2:

输入: 8

输出: 2

说明: 8 的平方根是 2.82842...,

  由于返回类型是整数，小数部分将被舍去。

要点：二分 或者 牛顿

class Solution {

    public int mySqrt(int x) {

        if(x==0) return 0;

        double C=x,x0=x;

        while(true){

            double x1=0.5\*(x0+C/x0);

            if(Math.abs(x1-x0)<1e-7) break;

            x0=x1;

        }

        return (int)x0;

    }

}

**32.搜素旋转排序数组**

给你一个整数数组 nums ，和一个整数 target 。

该整数数组原本是按升序排列，但输入时在预先未知的某个点上进行了旋转。（例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] ）。

请你在数组中搜索 target ，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

示例 1：

输入：nums = [4,5,6,7,0,1,2], target = 0

输出：4

示例 2：

输入：nums = [4,5,6,7,0,1,2], target = 3

输出：-1

示例 3：

输入：nums = [1], target = 0

输出：-1

要点：二分法，通过nums[0]和nums[mid]的值判断是在前半段序列还是后半段序列，进入每个序列后根据条件判断左右赋值。

class Solution {

    public int search(int[] nums, int target) {

        int n = nums.length;

        if (n == 0) {

            return -1;

        }

        if (n == 1) {

            return nums[0] == target ? 0 : -1;

        }

        int l = 0, r = n - 1;

        while (l <= r) {

            int mid = (l + r) / 2;

            if (nums[mid] == target) {

                return mid;

            }

            if (nums[0] <= nums[mid]) {

                if (nums[0] <= target && target < nums[mid]) {

                    r = mid - 1;

                } else {

                    l = mid + 1;

                }

            } else {

                if (nums[mid] < target && target <= nums[n - 1]) {

                    l = mid + 1;

                } else {

                    r = mid - 1;

                }

            }

        }

        return -1;

    }

}

**33.有效的数独**

判断一个 9x9 的数独是否有效。只需要根据以下规则，验证已经填入的数字是否有效即可。

数字 1-9 在每一行只能出现一次。

数字 1-9 在每一列只能出现一次。

数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

数独部分空格内已填入了数字，空白格用 '.' 表示。

要点：可以使用 box\_index = (row / 3) \* 3 + columns / 3，其中 / 是整数除法。

class Solution {

public boolean isValidSudoku(char[][] board) {

// init data

HashMap<Integer, Integer> [] rows = new HashMap[9];

HashMap<Integer, Integer> [] columns = new HashMap[9];

HashMap<Integer, Integer> [] boxes = new HashMap[9];

for (int i = 0; i < 9; i++) {

rows[i] = new HashMap<Integer, Integer>();

columns[i] = new HashMap<Integer, Integer>();

boxes[i] = new HashMap<Integer, Integer>();

}

// validate a board

for (int i = 0; i < 9; i++) {

for (int j = 0; j < 9; j++) {

char num = board[i][j];

if (num != '.') {

int n = (int)num;

int box\_index = (i / 3 ) \* 3 + j / 3;

// keep the current cell value

rows[i].put(n, rows[i].getOrDefault(n, 0) + 1);

columns[j].put(n, columns[j].getOrDefault(n, 0) + 1);

boxes[box\_index].put(n, boxes[box\_index].getOrDefault(n, 0) + 1);

// check if this value has been already seen before

if (rows[i].get(n) > 1 || columns[j].get(n) > 1 || boxes[box\_index].get(n) > 1)

return false;

}

}

}

return true;

}

}

**34.缺失的第一个正数**

给你一个未排序的整数数组，请你找出其中没有出现的最小的正整数。

示例 1:

输入: [1,2,0]

输出: 3

示例 2:

输入: [3,4,-1,1]

输出: 2

示例 3:

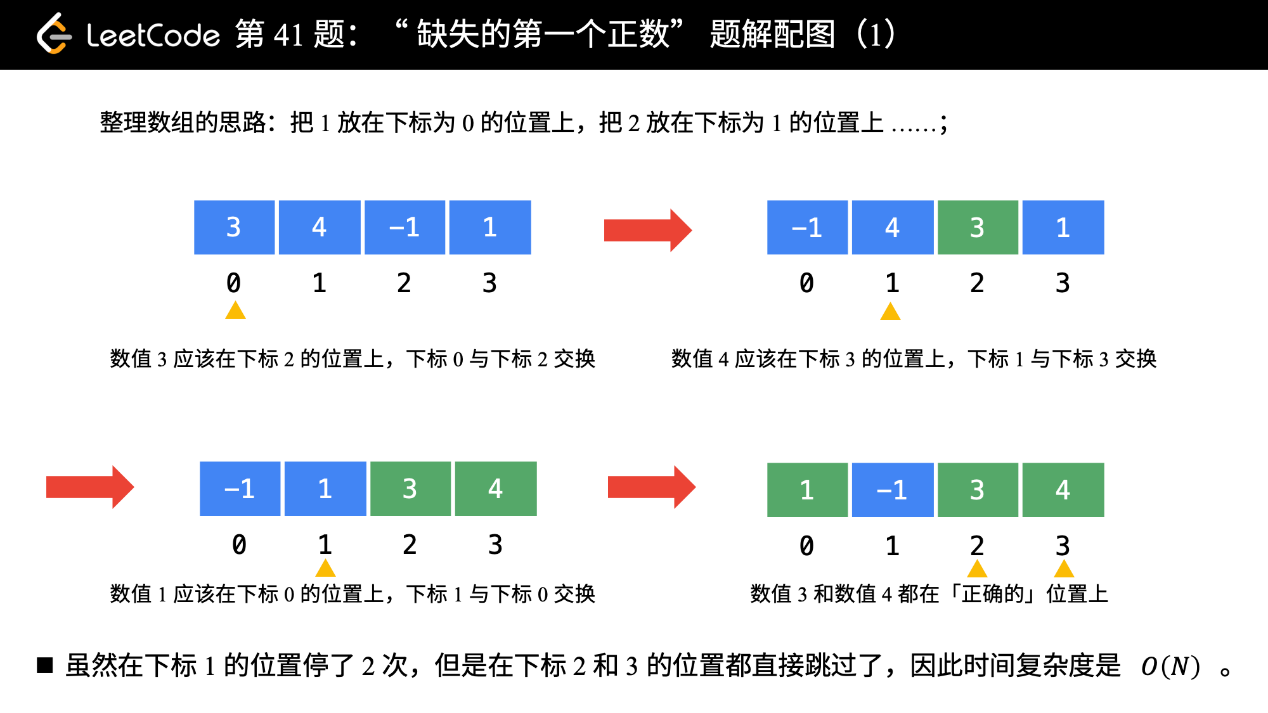
输入: [7,8,9,11,12]

输出: 1

提示：

你的算法的时间复杂度应为O(n)，并且只能使用常数级别的额外空间。

要点：



class Solution {

    public int firstMissingPositive(int[] nums) {

        int len = nums.length;

        for (int i = 0; i < len; i++) {

            while (nums[i] > 0 && nums[i] <= len && nums[nums[i] - 1] != nums[i]) {

                // 满足在指定范围内、并且没有放在正确的位置上，才交换

                // 例如：数值 3 应该放在索引 2 的位置上

                swap(nums, nums[i] - 1, i);

            }

        }

        // [1, -1, 3, 4]

        for (int i = 0; i < len; i++) {

            if (nums[i] != i + 1) {

                return i + 1;

            }

        }

        // 都正确则返回数组长度 + 1

        return len + 1;

    }

    private void swap(int[] nums, int index1, int index2) {

        int temp = nums[index1];

        nums[index1] = nums[index2];

        nums[index2] = temp;

    }

}

**35.Pow(x,n)**

实现 pow(x, n) ，即计算 x 的 n 次幂函数。

示例 1:

输入: 2.00000, 10

输出: 1024.00000

示例 2:

输入: 2.10000, 3

输出: 9.26100

示例 3:

输入: 2.00000, -2

输出: 0.25000

解释: 2-2 = 1/22 = 1/4 = 0.25

要点：快速幂

class Solution {

public double myPow(double x, int n) {

if(x == 0.0f) return 0.0d;

long b = n;

double res = 1.0;

if(b < 0) {

x = 1 / x;

b = -b;

}

while(b > 0) {

if((b & 1) == 1) res \*= x;

x \*= x;

b >>= 1;

}

return res;

}

}