笔试题

```
public class Convert {
    private static char[] array =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();
    private static String numStr = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    public static String _10_to_N(long number, int N) {
        Stack<Character> stack = new Stack<>();
        StringBuilder res = new StringBuilder();
        while (number != 0) {
            stack.push(array[(int)(number % N)]);
            number /= N;
        }
        while (!stack.isEmpty()) {
            res.append(stack.pop());
        }
        return res.toString();
    }

    public static long N_to_10(String number, int N) {
        long base = 1;
        long res = 0;
        for(int i = number.length() - 1; i >= 0; i--) {
            char c = number.charAt(i);
            res += base * numStr.indexOf(c);
            base *= N;
        }
        return res;
    }
//第二题
public class tt {
    static String[] keep=
{"Janurary","Feburary","March","April","May","June","July","Augest",
            "September","October","November","December"};
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        double[] in=new double[12];
        double[] out=new double[12];
        for(int i=0;i<12;i++) {
            String l1 = sc.nextLine();
            String l2 = sc.nextLine();
            String[] s1 = l2.split(" ");
            for (String ss : s1) {
                int i1 = ss.indexOf(":");
                double v = Double.parseDouble(ss.substring(i1 + 2));
                if (ss.charAt(i1 + 1) == '+') in[i] += v;
                else out[i] -= v;
            }
        }
        int minindex=0;//支出最多的是哪个月
        int maxindex=0;//收入最多的是哪个月
        for(int i=1;i<12;i++){
            if(in[i]>in[maxindex]) maxindex=i;
            if(out[i]<out[minindex]) minindex=i;
          }
```

```
        for(int i=0;i<12;i++){
            System.out.print(keep[i]+":");
            if(in[i]+out[i]>=0.0) System.out.print("+");
            System.out.println(String.format("%.2f",(in[i]+out[i])));
        }
        System.out.println(keep[minindex]+" "+keep[maxindex]);
    }
}
```

前k个高频元素

```
//使用快排排序解这个题，根据出现次数来排序
//自己实现一个小顶堆
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
       HashMap<Integer,Integer> map=new HashMap<>();
       for(int num:nums){
           map.put(num,map.getOrDefault(num,0)+1);
       }
       List<int[]> list=new ArrayList<>();
       for(Map.Entry<Integer,Integer> entry:map.entrySet()){
          list.add(new int[]{entry.getKey(),entry.getValue()});
       }
       buildheap(list);
       for(int i=list.size()-1;i>=0;i--){
           Collections.swap(list,0,i);
           adjust(list,0,i);
       }
       int[] res=new int[k];
       for(int i=0;i<k;i++) res[i]=list.get(i)[0];
       return res;
    }
    private void buildheap(List<int[]> list){
        int len=list.size();
        for(int i=len/2-1;i>=0;i--){
            adjust(list,i,len);
        }
    }
    private void adjust(List<int[]> list,int parent,int length){
        int child=2*parent+1;
        while(child<length){
            if(child+1<length && list.get(child)[1]>list.get(child+1)[1])
child++;
            if(list.get(parent)[1]<list.get(child)[1]) break;
            else Collections.swap(list,child,parent);
            parent=child;
            child=2*child+1;
        }
    }
}
//使用已经有的小顶堆
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
       int[] res=new int[k];
       if(nums.length==1 && k==1) return new int[]{nums[0]};
```

```
        PriorityQueue<int[]> minheap=new PriorityQueue<>((i1,i2)->i1[1]-
i2[1]);
        HashMap<Integer,Integer> map=new HashMap<>();
        for(int num:nums){
            map.put(num,map.getOrDefault(num,0)+1);
        }
        for(Map.Entry<Integer,Integer> enrty:map.entrySet()){
            int value=enrty.getKey();
            int count=enrty.getValue();
            if(minheap.size()==k){
                if(minheap.peek()[1]<count){
                    minheap.poll();
                    minheap.offer(new int[]{value,count});
                }
            }else minheap.offer(new int[]{value,count});
        }
        for(int i=0;i<k;i++){
            res[i]=minheap.poll()[0];
        }
        return res;
    }
}
```

下一个排列

```
class Solution {
    public void nextPermutation(int[] nums) {
      int len=nums.length;
      if(len<=1) return;
      int i=len-2,j=len-1,k=len-1;
      while(i>=0 && nums[i]>=nums[j]){//找到第一个左边比右边小的位置i,
          i--;
          j--;
      }
      if(i>=0){//如果遍历完都没有找到，说明是[321这种情况，直接逆序操作，将其转换为最小
的排列；否则找到比nums[i]大的索引位置k;
          while(nums[i]>=nums[k]) k--;//找到第一个比i大的位置k,交换i,k
          swap(nums,i,k);//交换i与k的值；
      }
      reverse(nums,i+1,len-1);//然后将i后的值进行升序操作，之前是降序的；
    }
    private void swap(int[] nums,int i,int j){
        int temp=nums[i];
        nums[i]=nums[j];
        nums[j]=temp;
    }
    private void reverse(int[] nums,int i,int j){
        while(i<j){
            swap(nums,i,j);
            i++;
            j--;
        }
    }
}
```

最大子数组和

```
//返回具体的子序列：使用left与i来实时更新子序列的左右指针
public static List<Integer> maxsum(int[] nums){
        HashMap<Integer,int[]> map=new HashMap<>();
        int left=0;
        int sum=0;
        int res=nums[0];
        map.put(res,new int[]{nums[0]});
        for(int i=0;i<nums.length;i++){
            if(sum<0) {//sum<0的时候，说明这个子数组的左边界一直在移动，需要将left重新变为i
                sum=nums[i];
                left=i;
            }
            else {
                sum+=nums[i];
                if(sum>res){
                    res=sum;
                    map.put(res,new int[]{left,i});
                }
            }
        }
        int i=map.get(res)[0];
        int j=map.get(res)[1];
        List<Integer> list=new LinkedList<>();
        for(int m=i;m<=j;m++){
            list.add(nums[m]);
        }
        return list;
}
```

二叉树中和为某个值的路径是否存在

```
    public boolean hasPathSum (TreeNode root, int sum) {
        if(root==null) return false;
        return dfs(root,sum);
    }
    private boolean dfs(TreeNode root,int sum){
        if(root==null) return false;
        sum-=root.val;
        if(root.left==null && root.right==null && sum==0) return true;
        return dfs(root.left,sum) || dfs(root.right,sum);
    }
```

二叉树的最大路径和

```
class Solution {//找每个分支的最大值并且实时更新每个跨根节点的路径和最大值；
    int res=Integer.MIN_VALUE;
    public int maxPathSum(TreeNode root) {
        if(root==null) return 0;
        dfs(root);
        return res;
    }
    private int dfs(TreeNode root){
        if(root==null) return 0;
```

```
        int leftmax=Math.max(0,dfs(root.left));
        int rightmax=Math.max(0,dfs(root.right));
        res=Math.max(res,root.val+leftmax+rightmax);
        return Math.max(root.val+leftmax,root.val+rightmax);
    }
}
```

将二叉树转换为链表

```
//通过二叉搜索的中序遍历方式，可以写三种不同的转换方法：
//1.递归 2.迭代 3.mirror但要记住这是双向链表，要记得头尾节点的连接
//中序遍历转换：将二叉树转换为链表
public class Solution {
    public TreeNode Convert(TreeNode pRootOfTree) {
        TreeNode root=pRootOfTree;
        TreeNode pre=null;
        TreeNode head=null;
        while(root!=null){
            if(root.left!=null){
                TreeNode pree=root.left;
                while(pree.right!=null) pree=pree.right;
                TreeNode temp=root;
                pree.right=root;
                root=root.left;
                temp.left=null;
            }else {
                if(pre==null) head=root;
                else pre.right=root;//使用left与right来维护链表的前后指针
                root.left=pre;
                pre=root;
                root=root.right;
            }
        }
        return head;
    }
}
//先序遍历转换：注意如果要求输出转换后的链表，那就需要用pre指针和head来维护他的链表的头结点，pre指的是遍历到node的前一个节点，若均将链表翻转成链表之后，接下来需要进行链表的zhi'xia
class Solution {
    public void flatten(TreeNode root) {
        while(root!=null){
            if(root.left!=null){
                TreeNode pre=root.left;
                while(pre.right!=null) pre=pre.right;
                pre.right=root.right;
                root.right=root.left;
                root.left=null;
            }else{
                root=root.right;
            }
        }
    }
}
```

判断是否为二叉树搜索树

```
public class Solution {//递归方法，还有一种是迭代方法
    public boolean isValidBST (TreeNode root) {
        long min=Long.MIN_VALUE,max=Long.MAX_VALUE;
        return dfs(root,min,max);
    }
    private boolean dfs(TreeNode node,long min,long max){
        if(node==null) return true;
        if(node.val<=min || node.val>=max) return false;
        return dfs(node.left,min,node.val) && dfs(node.right,node.val,max);
    }
}
```

判断一棵树是否为完全二叉树:判断左右不双全的节点是否为叶子节点

```
    public boolean isCompleteTree (TreeNode root) {
        if(root==null) return true;
        Queue<TreeNode> queue=new LinkedList<>();
        queue.offer(root);
        boolean flag=false;
        //前面的节点已经标记成叶子节点了，那么下一个节点应该不能有子节点了
        while(!queue.isEmpty()){
            TreeNode temp=queue.poll();
            TreeNode left=temp.left;
            TreeNode right=temp.right;
            if((flag && !(left==null && right==null)) || (left==null &&
right!=null)) return false;
            if(temp.left!=null) queue.offer(temp.left);
            if(temp.right!=null) queue.offer(temp.right);
            if(left==null || right==null) flag=true;
        }
        return true;
    }
```

判断是不是平衡二叉树

```
public class Solution {
    public boolean IsBalanced_Solution(TreeNode root) {
        if(root==null) return true;
        int L=dfs(root.left);
        int R=dfs(root.right);
        if(Math.abs(L-R)>1) return false;
        return IsBalanced_Solution(root.left) &&
IsBalanced_Solution(root.right);
    }
    private int dfs(TreeNode root){
        if(root==null) return 0;
        return Math.max(dfs(root.left),dfs(root.right))+1;
    }
}
```

二叉搜索树的公共祖先：公共祖先即找到

```
public class Solution {
    public int lowestCommonAncestor (TreeNode root, int p, int q) {
        if(root==null) return 0;
        if(root.val==p || root.val==q) return root.val;
        if(p>q){
            int temp=p;
            p=q;
            q=temp;
        }
        if(root.val<p) return lowestCommonAncestor(root.right,root.val,q);
        else if(root.val>q) return
lowestCommonAncestor(root.left,p,root.val);
        else return root.val;
    }
}
```

二叉树的最近公共祖先

```
public class Solution {
    public int lowestCommonAncestor (TreeNode root, int o1, int o2) {
        if(root==null) return 0;
        if(root.val==o1 || root.val==o2) return root.val;
        int left=lowestCommonAncestor(root.left,o1,o2);
        int right=lowestCommonAncestor(root.right,o1,o2);
        if(left==0 && right==0) return 0;
        if(left==0) return right;
        if(right==0) return left;
        return root.val;
    }
}
```

序列化二叉树,序列化：将二叉树转换为字符串，反序列化：将字符串转换为二叉树;

```
import java.util.*;
public class Solution {
    String Serialize(TreeNode root) {
        if(root==null) return "None,";
        return dfsserialize(root,"");
    }
    TreeNode Deserialize(String str) {
       String[] array=str.split(",");
       List<String> list=new ArrayList<>(Arrays.asList(array));
       return dfsDeserialize(list);
    }
    private String dfsserialize(TreeNode root,String s){
        if(root==null) s+="None,";
        else{
            s+=String.valueOf(root.val)+",";
            s=dfsserialize(root.left,s);
            s=dfsserialize(root.right,s);
        }
        return s;
```

```
    }
    private TreeNode dfsDeserialize(List<String> list){
        if(list.get(0).equals("None")){
            list.remove(0);
            return null;
        }else{
            TreeNode root=new TreeNode(Integer.valueOf(list.get(0)));
            list.remove(0);
            root.left=dfsDeserialize(list);
            root.right=dfsDeserialize(list);
            return root;
        }
    }
}
```

不同的子序列

```
class Solution {
    public int numDistinct(String s, String t) {
        int len1=s.length(),len2=t.length();
        int[][] dp=new int[len1+1][len2+1];
        for(int i=0;i<=len1;i++){
            for(int j=0;j<=len2;j++){
                if(i<j){
                    dp[i][j]=0;
                    continue;
                }
                if(i==0 || j==0){
                    dp[i][j]=1;
                    continue;
                }
                if(s.charAt(i-1)==t.charAt(j-1)) dp[i][j]=dp[i-1][j-1]+dp[i-1][j];
                else dp[i][j]=dp[i-1][j];
            }
        }
        return dp[len1][len2];
    }
}
```

二叉搜索树的第k大节点：

```
//递归：二叉搜索树的特点是：根的左子树的值均小于根，右子树的值均大于根
//所以可以根据遍历的方式，从大到小的方式进行遍历，第k次遍历的那个节点就是结结果
//二叉搜索树从小到大的方式：中序遍历-左根右
//从大到小：中序遍历的倒序：右根左
//时间为o(n),最坏情况需要遍历所有的节点；
//空间为o(n),最坏情况就是树退化成链表的情况(全部为右节点)，那么就额外占用o(N)的栈空间；
class Solution {
    int res,k;
    public int kthLargest(TreeNode root, int k) {
      this.k=k;
      this.res=0;
      dfs(root);
      return res;
    }
    private void dfs(TreeNode root){
```

```
        if(root==null || k==0) return;
        dfs(root.right);
        if(--k==0) res=root.val;
        dfs(root.left);
    }
}
```

最长公共子序列

```
class Solution {
    public int longestCommonSubsequence(String text1, String text2) {
        int t1=text1.length(),t2=text2.length();
        int[][] dp=new int[t1+1][t2+1];
        for(int i=0;i<=t1;i++){
            for(int j=0;j<=t2;j++){
                if(i==0 || j==0){
                    dp[i][j]=0;
                    continue;
                }
                if(text1.charAt(i-1)==text2.charAt(j-1)) dp[i][j]=dp[i-1][j-1]+1;
                else dp[i][j]=Math.max(dp[i-1][j],dp[i][j-1]);
            }
        }
        return dp[t1][t2];
    }
}
```

滑动窗口最大值

```
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
       Deque<Integer> deque=new LinkedList<>();
       int len=nums.length;
       int[] res=new int[len-k+1];
       for(int i=1-k,j=0;i<0;i++,j++){
           while(!deque.isEmpty() && deque.peekLast()<nums[j]){
               deque.pollLast();
           }
           deque.offerLast(nums[j]);
       }
       for(int i=0,j=k-1;j<len;i++,j++){
           if(i>0 && !deque.isEmpty() && nums[i-1]==deque.peekFirst()) deque.pollFirst();
           while(!deque.isEmpty() && nums[j]>deque.peekLast()) deque.pollLast();
           deque.offerLast(nums[j]);
           res[i]=deque.peekFirst();
       }
       return res;
    }
}
```

快乐数

```java
//快乐数：每个数字的下一位都是当前数字每位数字的平方和组成的
//最后的结果有两种可能：
//要么最终的结果为1，那他就是快乐数
//要么最终一直在不断的循环，导致到不了1，结束不了
//使用set集合来判断是否存在环形，存在的话就不是快乐数
class Solution {
    public boolean isHappy(int n) {
        HashSet<Integer> set=new HashSet<>();
        while(n!=1){
            int num=0;
            while(n!=0){
                num+=Math.pow(n%10,2);
                n/=10;
            }
            if(!set.add(num)) return false;
            n=num;
        }
        return true;
    }
}
```

整数反转:Integer.MAX_VALUE的最大值为2的31次方-1，最后一个数字为7，而Integer.MIN_VALUE的最后一个数字为-8；

```java
class Solution {
    public int reverse(int x) {
       int res=0,temp=0;
       while(x!=0){
           temp=x%10;
           if(res>Integer.MAX_VALUE/10 || (res==Integer.MAX_VALUE &&
temp>7)){
               return 0;
           }
           if(res<-Integer.MIN_VALUE/10 || (res==-Integer.MIN_VALUE &&
temp<-8)){
               return 0;
           }
           res=res*10+temp;
           x/=10;
       }
       return res;

    }
}
```

丑数：因子只含质因子2,3,5的数字, 输入10，输出12；

我们把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n 个丑数。1也是丑数

```java
//丑数：只包含质因子2、3、5
//质数：只有1与本身作为因数：例如1，2，3，5，7，11，13，17，19....
//质因子：能整除整数的数，并且也是质数的因子就是质因子
//例如18=3*6，3是质因子而6不是质因子
```

```java
//18=2*3*3,这里的2，3都是质因子
//本题需要找出从小到大顺序的第n个丑数
//第n个丑数的规律f(n)：
//找到f(a)*2,f(b)*3,f(c)*5的最小值=f(n)
//使用动态规划：d[i]代表第i-1个丑数
//初始化a==b==c==0;
//第一个丑数为1->dp[0]=1;
//若下一个丑数是dp[a]*2,a++;否则b++,否则c++
//时间o(n),空间o(n)
class Solution {
    public int nthUglyNumber(int n) {
        int[] dp=new int[n+1];
        dp[0]=1;
        int a=0,b=0,c=0;//初始化最初的三个索引为0，就是第一个丑数
        for(int i=1;i<n;i++){
            int n1=dp[a]*2,n2=dp[b]*3,n3=dp[c]*5;
            dp[i]=Math.min(Math.min(n1,n2),n3);
            if(dp[i]==n1) a++;
            if(dp[i]==n2) b++;
            if(dp[i]==n3) c++;
        }
        return dp[n-1];
    }
}
```

岛屿的最大面积

```java
class Solution {
    int[][] grid;
    int res=0;
    public int maxAreaOfIsland(int[][] grid) {
        this.grid=grid;
        int row=grid.length,col=grid[0].length;
        boolean[][] visited=new boolean[row][col];
        for(int i=0;i<row;i++){
            for(int j=0;j<col;j++){
                if(grid[i][j]==1){
                    int sum=0;
                    dfs(i,j,visited,sum);
                    res=Math.max(sum,res);
                }
            }
        }
        return res;
    }
    private void dfs(int i,int j,boolean[][] visited,int sum){
        if(i<0 || i>=grid.length || j<0 || j>=grid[0].length || grid[i][j]==0 || visited[i][j]==true) return;
        visited[i][j]=true;
        sum++;
        dfs(i-1,j,visited,sum);
        dfs(i+1,j,visited,sum);
        dfs(i,j-1,visited,sum);
        dfs(i,j+1,visited,sum);
    }
}
```

岛屿的路径

```
//最后返回的岛屿数量==哈希j
class Solution {
    public int numDistinctIslands(int[][] grid) {
        HashSet<String> set = new HashSet<>();
        for(int i = 0;i < grid.length;i++){
            for(int j = 0;j < grid[0].length;j++){
                if(grid[i][j]==0){
                    continue;
                }
                StringBuilder sb = new StringBuilder();
                dfs(grid,i,j,sb);
                set.add(sb.toString());
            }
        }
        return set.size();
    }
    public void dfs(int[][] grid,int i, int j, StringBuilder sb){
        if(i < 0 || i>=grid.length || j<0 || j>=grid[0].length ||grid[i][j]!=1){
            return;
        }
        //走到1就把当前的1清除掉，然后开始遍历下一个1，只能走相邻的1，只有相邻的1是同一个岛屿
        grid[i][j] = 0;
        dfs(grid,i+1,j,sb.append("d"));
        dfs(grid,i,j+1,sb.append("r"));
        dfs(grid,i-1,j,sb.append("u"));
        dfs(grid,i,j-1,sb.append("l"));
    }
}
```

岛屿的最大周长

```
class Solution {
    public int islandPerimeter(int[][] grid) {
       int row=grid.length;
       int col=grid[0].length;
       int res=0;
       int[][] dir=new int[][]{{1,0},{-1,0},{0,-1},{0,1}};
       for(int i=0;i<row;i++){
           for(int j=0;j<col;j++){
               if(grid[i][j]==1){
                   for(int[] d:dir){
                       int x=i+d[0];
                       int y=j+d[1];
                       if(x==-1 || x==row || y==-1 || y==col) res++;
                       if(x>=0 && x<row && y>=0 && y<col && grid[x][y]==0) res++;
                   }
               }
           }
       }
       return res;
    }
}
```

寻找数组的峰值

```
    public int findPeakElement (int[] nums) {
        if(nums.length==0) return -1;
        int left=0,right=nums.length-1;
        while(left<right){
            int m=left+(right-left)/2;
            if(nums[m]<nums[m+1]) left=m+1;
            else if(nums[m]>nums[m+1]) right=m;
        }
        return left;
    }
```

构建乘积数组

```
class Solution {
    public int[] constructArr(int[] a) {
       if(a.length==0) return new int[0];
       int[] b=new int[a.length];
       //下三角
       b[0]=1;
       for(int i=1;i<a.length;i++){
          b[i]=b[i-1]*a[i-1];
       }
       //上三角
       int temp=1;
       for(int i=a.length-2;i>=0;i--){
           temp*=a[i+1];
           b[i]*=temp;
       }
       return b;
    }
}
```

字符串转换为整数

```
public class Solution {

    public int myAtoi(String str) {
        int len = str.length();
        // str.charAt(i) 方法回去检查下标的合法性，一般先转换成字符数组
        char[] charArray = str.toCharArray();

        // 1、去除前导空格
        int index = 0;
        while (index < len && charArray[index] == ' ') {
            index++;
        }

        // 2、如果已经遍历完成（针对极端用例 "       "）
        if (index == len) {
            return 0;
        }

        // 3、如果出现符号字符，仅第 1 个有效，并记录正负
```

```
        int sign = 1;
        char firstChar = charArray[index];
        if (firstChar == '+') {
            index++;
        } else if (firstChar == '-') {
            index++;
            sign = -1;
        }

        // 4、将后续出现的数字字符进行转换
        // 不能使用 long 类型，这是题目说的
        int res = 0;
        while (index < len) {
            char currChar = charArray[index];
            // 4.1 先判断不合法的情况
            if (currChar > '9' || currChar < '0') {
                break;
            }

            // 题目中说：环境只能存储 32 位大小的有符号整数，因此，需要提前判：断乘以 10 以后是否越界
            if (res > Integer.MAX_VALUE / 10 || (res == Integer.MAX_VALUE / 10 && (currChar - '0') > Integer.MAX_VALUE % 10)) {
                return Integer.MAX_VALUE;
            }
            if (res < Integer.MIN_VALUE / 10 || (res == Integer.MIN_VALUE / 10 && (currChar - '0') > -(Integer.MIN_VALUE % 10))) {
                return Integer.MIN_VALUE;
            }

            // 4.2 合法的情况下，才考虑转换，每一步都把符号位乘进去
            res = res * 10 + sign * (currChar - '0');
            index++;
        }
        return res;
    }
```

从中序和后序重建二叉树

```
class Solution {
    Map<Integer, Integer> map = new HashMap<>();
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        int n = inorder.length;
        // 将中序遍历放到map中
        for (int i = 0; i < n; ++i) {
            map.put(inorder[i], i);
        }
        return myBuildTree(inorder, postorder, 0, n - 1, 0, n - 1);
    }

    public TreeNode myBuildTree(int[] inorder, int[] postorder, int inorder_left, int inorder_right, int postorder_left, int postorder_right) {
        if (inorder_left > inorder_right) {
            return null;
        }
        // 根节点在后序遍历中的下标
        int postorder_root = postorder_right;
```

```
        // 根节点在中序遍历中的根节点
        int inorder_root = map.get(postorder[postorder_root]);
        // 左子树的长度
        int size_left_subtree = inorder_root - inorder_left;
        // 建立根节点
        TreeNode root = new TreeNode(postorder[postorder_root]);
        root.left = myBuildTree(inorder, postorder, inorder_left,
inorder_root - 1, postorder_left, postorder_left + size_left_subtree - 1);
        root.right = myBuildTree(inorder, postorder, inorder_root + 1,
inorder_right, postorder_left + size_left_subtree, postorder_right - 1);
        return root;
    }
}
```

跳跃游戏III：两个方向，左右两个方向

```
class Solution {
    public boolean canReach(int[] arr, int start) {
        int n = arr.length;
        //防止进入死循环
        boolean[] visited = new boolean[n+1];
        Queue<Integer> queue = new LinkedList<>();
        queue.add(start);
        visited[start] = true;
        while(!queue.isEmpty()){
            int tmp = queue.poll();
            //两个方向
            int left = tmp - arr[tmp],right = tmp + arr[tmp];
            if(left >= 0){
                if(arr[left] == 0) return true;
                if(!visited[left]){
                    visited[left] = true;
                    queue.add(left);
                }
            }
            if(right < n){
                if(arr[right] == 0) return true;
                if(!visited[right]){
                    visited[right] = true;
                    queue.add(right);
                }
            }
        }
        return false;
    }
}
```

对角线遍历

```
public static int[] findDiagonalOrder(int[][] matrix) {
        if (matrix.length == 0) {
            return new int[0];
        }
        int row = matrix.length;
        int col = matrix[0].length;
        int[] answer = new int[row * col];
        int count = row + col - 1;//需要转换反向的次数，方向分别为右上和左下，
```

```
        int m = 0;//m,n分别为横纵坐标
        int n = 0;
        int answerIndex = 0;
        for (int i = 0; i < count; i++) {
            if (i % 2 == 0) {//转换方向
                while (m >= 0 && n < col) {
                    answer[answerIndex] = matrix[m][n];
                    answerIndex++;
                    m--;
                    n++;
                }
                if (n < col) {//到了要转换方向的边界了，开始注意坐标的变化，边界值的改变
                    m++;
                } else {
                    m = m + 2;
                    n--;
                }
            } else {
                while (m < row && n >= 0) {
                    answer[answerIndex] = matrix[m][n];
                    answerIndex++;
                    m++;
                    n--;
                }
                if (m < row) {
                    n++;
                }else{
                    m--;
                    n=n+2;
                }
            }
        }
        return answer;
    }
```

最佳买卖股票时期,只能两次购买和卖出股票的操作，记录可以得到的利润最大值

```
 public int maxProfit (int[] prices) {
     5个状态 0： 未操作 1 第一次购买持有股票 2 第一次卖出，不持有股票 3 第二次购买持有股票 4 第二次卖出不持有股票

        int len = prices.length;
        if (prices == null || len == 0){
            return  0 ;
        }
        int[][] dp = new int[len][5];
        dp[0][0] = 0;
        dp[0][1] = - prices[0];
        dp[0][3] = -prices[0];
        for (int i = 1; i <prices.length ; i++) {
            dp[i][0] = dp[i-1][0];
            dp[i][1] = Math.max(dp[i - 1][1], dp[i - 1][0] - prices[i]);
            dp[i][2] = Math.max(dp[i - 1][2], dp[i - 1][1] + prices[i]);
            dp[i][3] = Math.max(dp[i - 1][3], dp[i - 1][2] - prices[i]);
            dp[i][4] = Math.max(dp[i - 1][4], dp[i - 1][3] + prices[i]);
        }
```

```
        return dp[prices.length - 1][4];
    }
```

最佳股票收益含冷冻期

```
class Solution {
    public int maxProfit(int[] prices) {
        int len=prices.length;
        //0表示持有股票的最大收益，1表示处于冷冻期，2表示既没有进入冷冻期也没有股票
        int[][] dp=new int[len][3];
        dp[0][0]=-prices[0];
        for(int i=1;i<len;i++){
            dp[i][0]=Math.max(dp[i-1][0],dp[i-1][2]-prices[i]);
            dp[i][1]=dp[i-1][0]+prices[i];
            dp[i][2]=Math.max(dp[i-1][2],dp[i-1][1]);
        }
        return Math.max(dp[len-1][0],Math.max(dp[len-1][1],dp[len-1][2]));
    }
}
```

正则表达式

```
import java.util.*;
public class Solution {
    public boolean match (String str, String pattern) {
       int slen=str.length(),plen=pattern.length();
       boolean[][] dp=new boolean[slen+1][plen+1];
       for(int i=0;i<=slen;i++){
           for(int j=0;j<=plen;j++){
               if(j==0) dp[i][j]=i==0;
               else{
                   if(pattern.charAt(j-1)!='*'){
                      if(i>0 && (str.charAt(i-1)==pattern.charAt(j-1) ||
pattern.charAt(j-1)=='.')){
                         dp[i][j]=dp[i-1][j-1];
                      }
                   }else{
                       if(j>=2) dp[i][j] |= dp[i][j-2];
                       if(i>=1 && j>=2 && (str.charAt(i-
1)==pattern.charAt(j-2) || pattern.charAt(j-2)=='.')){
                          dp[i][j] |=dp[i-1][j];
                       }
                   }
               }
           }
       }
       return dp[slen][plen];
    }
}
```

最长回文子串

```
import java.util.*;
public class Solution {
    public int getLongestPalindrome (String A) {
        int res=Integer.MIN_VALUE;
```

```
        int len=A.length();
        for(int i=0;i<len;i++){
            for(int j=0;i-j>=0 && i+j<len;j++){
                if(A.charAt(i-j)==A.charAt(i+j)){
                    int temp=2*j+1;
                    res=Math.max(temp,res);
                }else break;
            }
        }
        if(len>1){
            for(int i=0;i+1<len;i++){
                for(int j=0;i-j>=0 && i+j+1<len;j++){
                    if(A.charAt(i-j)==A.charAt(i+j+1)){
                        int temp=2*j+2;
                        res=Math.max(res,temp);
                    }else break;
                }
            }
        }
        return res;
    }
}
```

单词拆分:s = "leetcode", wordDict = ["leet", "code"]

```
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
       HashSet<String> set=new HashSet<>(wordDict);
       int len=s.length();
       boolean[] dp=new boolean[len+1];
       dp[0]=true;
       for(int i=1;i<=len;i++){
           for(int j=0;j<i;j++){
               if(dp[j] && set.contains(s.substring(j,i))){
                   dp[i]=true;
                   break;
               }
           }
       }
       return dp[len];
    }
}
```

回文子串

```
class Solution {
    public int countSubstrings(String s) {
       int len=s.length();
       int n=2*len-1;
       int res=0;
       for(int i=0;i<n;i++){
           int L=i/2,R=i%2+L;
           while(L>=0 && R<len && s.charAt(L)==s.charAt(R)){
               res++;
               L--;
               R++;
           }
```

```
        }
        return res;
    }
}
```

目标和:向数组中的每个整数前添加 `'+'` 或 `'-'` ，然后串联起所有整数，可以构造一个 表达式 ：

```
class Solution {
    public int findTargetSumWays(int[] nums, int target) {
        int sum=0;
        int len=nums.length;
        for(int num:nums){
            sum+=num;
        }
        if(sum<target || (sum-target)%2!=0) return 0;
        int neg=(sum-target)/2;
        int[][] dp=new int[len+1][neg+1];
        dp[0][0]=1;
        for(int i=1;i<=len;i++){
            int num=nums[i-1];
            for(int j=0;j<=neg;j++){
                dp[i][j]=dp[i-1][j];
                if(j>=num) dp[i][j]+=dp[i-1][j-num];
            }
        }
        return dp[len][neg];
    }
}
```

和为k的子数组：前缀和

```
//前缀和
//当前位置的和sum[i]-sum[i-1]==k;
//说明i与i-1之间的的节点值和为k
//由于前缀和都为sum[i-1]的节点可以有多个，所以我们使用哈希表存储每个前缀和的个数
class Solution {
    public int subarraySum(int[] nums, int k) {
      HashMap<Integer,Integer> map=new HashMap<>();
      map.put(0,1);
      int count=0;
      int sum=0;
      for(int num:nums){
          sum+=num;
          if(map.containsKey(sum-k)) count+=map.getOrDefault(sum-k,0);
          map.put(sum,map.getOrDefault(sum,0)+1);
      }
      return count;
    }
}
```

字符串解码：s = "3[a]2[bc]"-----"aaabcbc"

```
class Solution {
    public String decodeString(String s) {
        Stack<Integer> multi_stack=new Stack<>();
        Stack<String> str_stack=new Stack<>();
```

```
        int multi=0;
        StringBuffer res=new StringBuffer();
        for(char c:s.toCharArray()){
            if(c>='0' && c<='9') multi=multi*10+c-'0';
            else if(c=='['){
                multi_stack.push(multi);
                str_stack.push(res.toString());
                multi=0;
                res=new StringBuffer();
            }else if(c==']'){
                int num=multi_stack.pop();
                StringBuffer temp=new StringBuffer();
                for(int i=0;i<num;i++) temp.append(res);
                res=new StringBuffer(str_stack.pop()+temp);
            }else res.append(c);
        }
        return res.toString();
    }
}
```

二叉树的直径

```
class Solution {
    int res;
    public int diameterOfBinaryTree(TreeNode root) {
       if(root==null) return 0;
       dfs(root);
       return res-1;
    }
    private int dfs(TreeNode root){
        if(root==null) return 0;
        int left=dfs(root.left);
        int right=dfs(root.right);
        res=Math.max(res,left+right+1);
        return Math.max(left,right)+1;
    }
}
```

分割等和子集

```
class Solution {
    public boolean canPartition(int[] nums) {
       int len=nums.length;
       int sum=0,maxnum=Integer.MIN_VALUE;
       for(int num:nums){
           sum+=num;
           maxnum=Math.max(num,maxnum);
       }
       int target=sum/2;
       if(target<maxnum || sum%2!=0) return false;
       boolean[][] dp=new boolean[len+1][target+1];
       dp[0][0]=true;
       for(int i=1;i<=len;i++){
           int num=nums[i-1];
           for(int j=0;j<=target;j++){
               dp[i][j]=dp[i-1][j];
               if(num==j){
```

```
                    dp[i][j]=true;
                    continue;
                }else if(j>num) dp[i][j] |= dp[i-1][j-num];
            }
        }
        return dp[len][target];
    }
}
```

课程表：先修课程->当前课程

```
class Solution {
    List<List<Integer>> list=new ArrayList<>();
    int[] visited;
    boolean res;
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        visited=new int[numCourses];
        for(int i=0;i<numCourses;i++){
            list.add(new ArrayList<>());
        }
        for(int[] p:prerequisites){
            list.get(p[1]).add(p[0]);
        }
        res=true;
        for(int i=0;i<numCourses;i++){
            if(visited[i]==0){
                dfs(i);
            }
        }
        return res;
    }
    private void dfs(int u){
        visited[u]=1;
        for(int v:list.get(u)){
            if(visited[v]==1){
                res=false;
                return;
            }else if(visited[v]==0){
                visited[v]=1;
                dfs(v);
                if(res==false) return;
            }
        }
        visited[u]=2;
    }
}
```

乘积最大数组

```
class Solution {
    public int maxProduct(int[] nums) {
        int res=Integer.MIN_VALUE;
        int min=1,max=1;
        for(int num:nums){
            if(num<0){
                int temp=max;
                max=min;
```

```
                min=temp;
            }
            max=Math.max(num,max*num);
            min=Math.min(num,min*num);
            res=Math.max(res,max);
        }
        return res;
    }
}
```

打家劫舍III：二叉树

```
class Solution {
    public int rob(TreeNode root) {
        if(root==null) return 0;
        int[] res=dfs(root);
        return Math.max(res[0],res[1]);//根节点偷或者不偷
    }
    private int[] dfs(TreeNode root){
        if(root==null) return new int[2];
        int[] res=new int[2];
        int[] left=dfs(root.left);
        int[] right=dfs(root.right);
        res[0]=left[1]+right[1]+root.val;
        res[1]=Math.max(left[0],left[1])+Math.max(right[0],right[1]);
        return res;
    }
}
```

前缀树

```
class Trie {
    Trie[] trie;
    boolean isend;
    public Trie() {
      trie=new Trie[26];
      isend=false;
    }

    public void insert(String word) {
      Trie node=this;
      for(char c:word.toCharArray()){
          if(node.trie[c-'a']==null){
            node.trie[c-'a']=new Trie();
          }
          node=node.trie[c-'a'];
      }
      node.isend=true;
    }

    public boolean search(String word) {
      Trie node=this;
      for(char c:word.toCharArray()){
           if(node.trie[c-'a']==null) return false;
           else node=node.trie[c-'a'];
      }
      return node.isend;
```

```
    }

    public boolean startsWith(String prefix) {
      Trie node=this;
      for(char c:prefix.toCharArray()){
          if(node.trie[c-'a']==null) return false;
          else node=node.trie[c-'a'];
      }
      return true;
    }
}
```

最大正方形

```
class Solution {
    public int maximalSquare(char[][] matrix) {
      int m=matrix.length;
      int n=matrix[0].length;
      int[][] dp=new int[m][n];
      int res=Integer.MIN_VALUE;
      if(matrix[0][0]=='1') dp[0][0]=1;
      else dp[0][0]=0;
      for(int i=0;i<m;i++){
          for(int j=0;j<n;j++){
             if(matrix[i][j]=='1'){
                 if(i==0 || j==0) dp[i][j]=1;
                 else dp[i][j]=Math.min(dp[i-1][j],Math.min(dp[i][j-1],dp[i-1][j-1]))+1;
                 res=Math.max(res,dp[i][j]);
             }
          }
      }
      return res==Integer.MIN_VALUE?0:res*res;
    }
}
```

搜索矩阵

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
       int i=0,j=matrix[0].length-1;//从最右上角开始找，第一行最后一列
       while(i<matrix.length && j>=0){
           if(matrix[i][j]>target) j=j-1;
           else if(matrix[i][j]<target) i=i+1;
           else return true;
       }
       return false;
    }
}
```

寻找重复数：nums = [1,3,4,2,2]---2;

```
//可以使用环形链表的思路解题：
//时间o(n),空间o(1);
class Solution {
    public int findDuplicate(int[] nums) {
```

```
        int slow=0,fast=0;
        slow=nums[slow];
        fast=nums[nums[fast]];
        //构建第一次相遇
        while(slow!=fast){
          //慢指针下一次跳到上一个值指代的索引处；
          slow=nums[slow];
          //快指针下一次跳到上一个值指代索引处值的索引；
          fast=nums[nums[fast]];//注意这里的快指针在数组中的表示；挺难想的；
        }
        //构建第二次相遇；
        int pre1=0;
        while(pre1!=slow){
            pre1=nums[pre1];
            slow=nums[slow];
        }
        return pre1;
    }
}
```

根据身高重建队列

```
class Solution {
    public int[][] reconstructQueue(int[][] people) {
        Arrays.sort(people,(i1,i2)->i1[0]==i2[0]?i1[1]-i2[1]:i2[0]-i1[0]);
        List<int[]> list=new ArrayList<>();
        for(int[] p:people){
            list.add(p[1],p);
        }
        return list.toArray(new int[list.size()][2]);
    }
}
```

零钱兑换

```
//动态规划
//使用数组dp找出金额为amout的最少硬币数量
class Solution {
    public int coinChange(int[] coins, int amount) {
        int[] dp=new int[amount+1];
        dp[0]=0;
        for(int i=1;i<=amount;i++){
            //每个金额的初始最小硬币数量
            int min=Integer.MAX_VALUE;
            for(int j=0;j<coins.length;j++){
                //不断的更新最小硬币数量
                if(i>=coins[j] && dp[i-coins[j]]<min) min=dp[i-coins[j]]+1;
            }
            dp[i]=min;//i金额的最小硬币数量为为min;
        }
        return dp[amount]==Integer.MAX_VALUE?-1:dp[amount];
    }
}
```

戳气球

```
//动态规划：
//时间o(n^3),空间o(n^2),空间存储气球数量；

//分别判断数组中第i个气球最后戳破所得到的硬币数量；
//这里越界的值为1，那么就可以将nums数组左右各+1;
//例如：[3，1，5，8]左右加+1=[1,3,1,5,8,1]
//eg:加入最后戳破5,索引位置为3，那么最大硬币数量为

//dp[0][3]开区间内戳破气球所获得的硬币数量与
//dp[3][len-1]开区间内戳破气球所获得的硬币与
//nums[3]*nums[0]*nums[len-1]三者的和；

//动态规划转移方程为：
//dp[i][j]开区间内最后戳破第k个气球所获得的硬币数量：
//dp[i][j]=Math.max(dp[i][j],dp[i][k]+dp[k][j]+nums[k]*nums[i]*nums[j]))
class Solution {
    public int maxCoins(int[] nums) {
        int len=nums.length+2;
        int[] temp=new int[len];
//左右各加1之后的数组，将越界的值存入数组方便后续不单独考虑数组越界的情况
        int[][] dp = new int[len][len];//dp数组保证能两个1中间的所有值
        temp[0]=temp[len-1]=1;//并且最左和最右各位1；
        for(int i=0;i<nums.length;i++){
            temp[i+1]=nums[i];
        }
        //要保证开区间内至少有一个数字，那么从dp左开区间i从len-3开始
        //dp右区间要从左区间+2的位置开始，确保i,j区间内至少一个数字
        //遍历i,j开区间内的每一个数字，求出若最后戳破这个气球可以得到的硬币
        for(int i=len-3;i>=0;i--){
            for(int j=i+2;j<len;j++){
              for(int k=i+1;k<j;k++){
                dp[i][j]=Math.max(dp[i][j],
                dp[i][k]+dp[k][j]+temp[k]*temp[i]*temp[j]);
              }
            }
        }
        return dp[0][len-1];
    }
}
```

多数元素

```java
//3.摩尔投票法：//将第一个数记为众数为1；那么不是1的数就记为0；进行加和，然后当和为0时，将当前值定义为众数，最后和不为0时最后那个众数就是结果；
//时间o(n).空间o(1);
class Solution {
    public int majorityElement(int[] nums) {
      if(nums.length==0) return 0;
      int res=nums[0],sum=0;
      for(int num:nums){
          if(sum==0) res=num;
          sum+=num==res?1:-1;
      }
      return res;
    }
}
```

ACM输入:链表

```java
    private static ListNodem creatlists(String[] s){
        if(s==null || s.length==0) return null;
        ListNodem dummy=new ListNodem(0);
        ListNodem cur=dummy;
        for(int i=0;i<s.length;i++){
            ListNodem temp=new ListNodem(Integer.parseInt(s[i]));
            cur.next=temp;
            cur=cur.next;
        }
        return dummy.next;
    }
    private static void print(ListNodem root){
        while(root!=null){
            System.out.print(root.val);
            if(root.next!=null){
                System.out.print("->");
            }
            root=root.next;
        }
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        String[] s1=sc.nextLine().split(",");
        String[] s2=sc.nextLine().split(",");
        ListNodem root1=creatlists(s1);
        ListNodem root2=creatlists(s2);
        ListNodem res=mergeTwoLists(root1,root2);
```

二叉树

```java
    private List<Integer> print(TreeNode4 root){
        List<Integer> res=new LinkedList<>();
        Queue<TreeNode4> queue=new LinkedList<>();
        queue.offer(root);
        while(!queue.isEmpty()){
            TreeNode4 temp=queue.poll();
            res.add(temp.val);
            if(temp.left!=null) queue.offer(temp.left);
            if(temp.right!=null) queue.offer(temp.right);
```

```
        }
        return res;
    }
    public static void main(String[] args){
        Scanner sc=new Scanner(System.in);
        int n1=sc.nextInt();
        int rootint1=sc.nextInt();
        TreeNode4[] T1=new TreeNode4[n1];
        for(int i=0;i<n1;i++){
            T1[i]=new TreeNode4(0);
        }
        for(int i=0;i<n1;i++){
            int value=sc.nextInt();
            int left=sc.nextInt();
            int right=sc.nextInt();
            T1[i].val=value;
            if(left!=0) T1[i].left=T1[left-1];
            if(right!=0) T1[i].right=T1[right-1];
        }
        int n2=sc.nextInt();
        int rootint2=sc.nextInt();
        TreeNode4[] T2=new TreeNode4[n2];
        for(int i=0;i<n2;i++){
            T2[i]=new TreeNode4(0);
        }
        for(int i=0;i<n2;i++){
            int value1=sc.nextInt();
            int left1=sc.nextInt();
            int right1=sc.nextInt();
            T2[i].val=value1;
            if(left1!=0) T2[i].left=T2[left1-1];
            if(right1!=0) T2[i].right=T2[right1-1];
        }
        mergeTreestest mer=new mergeTreestest();
        TreeNode4 res=mer.mergeTrees(T1[rootint1-1],T2[rootint2-1]);
        List<Integer> list=mer.print(res);
        System.out.println(list);
    }
}
//3 1 第一行第一个数字表示这个数共有几个节点，第二个数字表示第几个节点是根节点
//接下来输入3行，分别表示每个节点的值以及左右孩子的情况
//1 2 3 表示第一个节点值为1，左孩子第二个节点，右孩子第三个节点
//2 0 0 表示第二个节点为2，无左右孩子，0表示无孩子
//3 0 0 表示第三个节点为3，无左右孩子
```