//贪心算法

//LeetCode P455 分饼干问题

//思路：将两个数组排序，用最小值得饼干分给最小值的孩子，两个数组都循环，知道一个数组到最后

// 如果第i个孩子的值大于第i个饼干，则饼干向后移动，孩子不变，直到满足或循环结束；如果第i个孩子的值不大于第i个饼干的值，则两个都向后移动

int g [] = {4,5,6,1,3,4};

int s [] = {2,4,5,7,6,5,4};

int ch = 0, co = 0;

Arrays.sort(g);

Arrays.sort(s);

while (ch<g.length && co<s.length){

if (g[ch]<=s[co]){

ch++;

}

co++;

}

System.out.println(ch);

//LeetCode P 分糖果问题

//队列中值较高的分到的糖果要比旁边两个的多

//思路：因为最少要有一个，所以可以将最开始的数列值设为1，然后先从左向右循环一次，如果右边的值大于左边，则右边这个的糖果数量=左边糖果数量+1；

// 第二次循环是从右向左循环，如果左边的数值大于右边，且左边分到的糖果数不大于右边的分到的糖果数量，则左边这个的糖果数量=右边糖果数量+1

int ratings [] = {1,2,87,87,87,2,1};

int arr [] = new int[ratings.length];

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

arr[i] = 1 ;

}

for (int i=0 ; i<arr.length-1 ; i++){

if(ratings[i+1] > ratings[i])

arr[i+1] = arr[i]+1;

}

for (int i=arr.length-1 ; i>0 ; i--){

if (ratings[i-1]>ratings[i] && arr[i-1]<=arr[i])

arr[i-1] = arr[i]+1;

}

int sum = 0;

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

sum += arr[i];

}

System.out.println(sum);

}

//LeetCode P435 区域不重复问题，可以是求不重复区间，也可以抽象成一个领导要开会，可能一些会议的时间会重复，所以要退掉一些会议，求最多能参加的会议数量

//思路：一个N行2列的二维数组，先将他们按右边的值（interval[i][1]）进行升序排列，然后按行遍历，看有没有重复的值

int intervals [][] = {{1,2},{2,3},{3,4}};

if (intervals == null)

System.out.println("no");

Arrays.sort(intervals, new Comparator<int[]>() { //重写comparator构造

public int compare(int[] a, int[] b) { //a[1]-b[1]是升序 b[1]-a[1]是降序 ，[i]是按第i-1个数排序

return a[1] - b[1];

}

});

//Arrays.sort(intervals,(a,b)->a[1]>b[1]?1:-1); 可以这样写

int count = 1;

int a = intervals[0][1];

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

if(a <= intervals[i][0]){

a = intervals[i][1];

++count;

}

}

System.out.println(intervals.length-count);

//LeetCode P452 用最少的箭穿过气球 和上一题重复区间类似

//思路：求重复最多的区间，可以先将二维数组按右边界升序排序，这里排序要用Arrays.*sort*(points, (a,b) -> a[1] > b[1] ? 1 : -1);因为会有边界值问题[[-2147483646,-2147483645],[2147483646,2147483647]]这组数据，直接返回两个数相减会报错，以后用Arrays.*sort*(points, (a,b) -> a[1] > b[1] ? 1 : -1)这个方法排序。排好序后，遍历数组。如果一个区间的右边界大于后一个数的左边界，则说明重合。

一直往后遍历，直到不重合就加1，即用最少可以穿过所有气球。

**int** points [][] = {{10,16},{2,8},{1,6},{7,12}};  
Arrays.*sort*(points, (a,b) -> a[1] > b[1] ? 1 : -1);  
**int** count = 1;  
**int** a = points[0][1];  
**for** (**int** i=1 ; i<points.**length** ; i++){  
 **if**(a < points[i][0]){  
 ++count;  
 a = points[i][1];  
 }  
}  
System.***out***.println(count);

//P605 种花问题

//思路：找数组中的0的个数，用1隔开，每一段0的个数，然后可以种的数量和0的数量有一定给的关系

if(flowerbed.length == 0) return false;

if(flowerbed.length ==1 && flowerbed[0]==0 && n==1) return true;

String s = "";

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

s = s+flowerbed[i];

}

String [] res = s.split("1");

int [] a = new int [res.length];

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

a[i] = res[i].length();

}

int sum = 0;

if (!s.contains("1")){

if(s.length()%2 == 0)

sum =s.length()/2;

else

sum =s.length()/2 + 1;

}

else{

if (s.indexOf('1') != 0 ){

sum = sum+s.indexOf('1')/2;

a[0]=0;

}

if (s.lastIndexOf('1') != s.length()-1){

sum = sum + ((s.length()-1-s.lastIndexOf('1'))/2);

a[a.length-1]=0;

}

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

if (a[i] <=2)

continue;

else if(a[i]%2 == 0)

sum = sum + a[i]/2-1;

else

sum = sum + a[i]/2;

}

}

if(sum >= n) return true;

else return false;

//别人的代码，思路一样的

//用的数学归纳法，在中间的0，可以找到规律，在两边的0和在中间的0的规律不同，但是在左右两边都加一个0后，规律就变得相同

if (flowerbed == null || flowerbed.length == 0) return n == 0;

int countOfZero = 1; // 当前全0区段中连续0的数量，刚开始预设1个0，因为开头花坛的最左边没有花，可以认为存在一个虚无的0

int canPlace = 0; // 可以种的花的数量

for (int bed : flowerbed) {

if (bed == 0) { // 遇到0，连续0的数量+1

countOfZero++;

} else { // 遇到1，结算上一段连续的0区间，看能种下几盆花：(countOfZero-1)/2

canPlace += (countOfZero-1)/2;

if (canPlace >= n) return true;

countOfZero = 0; // 0的数量清零，开始统计下一个全0分区

}

}

// 最后一段0区还未结算：

countOfZero++; // 最后再预设1个0，因为最后花坛的最右边没有花，可以认为存在一个虚无的0

canPlace += (countOfZero-1)/2;

return canPlace >= n;

//用贪心算法

//遍历数组，看哪里可以种花就直接种，左边右边都不是1就可以种

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

if(n <= 0) { // 如果已经种够花了，可以提前返回true

return true;

}

if(flowerbed[i] == 1) { // 如果已经种过花了，则不能再种了

continue;

}

if(i > 0 && flowerbed[i - 1] == 1) { // 如果上一个格子已经种过花了，则当前这格不能种花

continue;

}

if(i < flowerbed.length - 1 && flowerbed[i + 1] == 1) { // 如果下一个格子已经种过花了，则当前这格不能种花

continue;

}

// 可以种花了，并且记录次数

flowerbed[i] = 1;

n--;

}

return n <= 0;