ACM

```
Scanner sc = new Scanner(System.in);
String str = sc.next();//获得空格前的第一个字符串
char c = sc.next().charAt(0);//获得空格前的第一个字符串的第一个值
String str = sc.nextLine();
String[] str = str.spilt(" ");//按空格划分
String str = sc.nextLine().toLowerCase();//转化字符串为小写
int num = sc.nextInt();//获得整型
Integer.parseInt(s[i]);//把字符串转成整型格式,来用
System.out.print(i + " ")//输出需要空格

//一般来说数组读取分两种
// 1.数组长度先给
Scanner sc = new Scanner(System.in);
int num = sc.nextInt();
int[] arr = new int[num];
for(int i = 0;i < num;i++)
{
    arr[i] = sc.nextInt();
}
//逗号 空格划分
Scanner sc = new Scanner(System.in);
String str = sc.nextLine();
String[] res = str.spilt(",");
int[] arr = new int[num];
for(int i = 0;i < res.length;i++)
{
    arr[i] = Integer.parseInt(res[i]);
}
```

ACM读入链表

```
class ListNode {
    int val;
    ListNode next;
    ListNode() {}
    ListNode(int val) {
        this.val = val;
    }
    ListNode(int val, ListNode next) {
        this.val = val;
        this.next = next;
    }
}
public class HuaWei {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] split = sc.nextLine().split(" ");
        int[] arr = new int[split.length];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = Integer.parseInt(split[i]);
        }
        ListNode dummy = new ListNode(-1);
```

```
        ListNode cur = dummy;
        for (int i = 0; i < arr.length; i++) {
            ListNode node = new ListNode(arr[i]);
            cur.next = node;
            cur = cur.next;
            if (i == arr.length - 1) {
                cur.next = null;
            }
        }
        ListNode node = removeElements(dummy.next, val);//这里实现函数
        while (node != null) {
            System.out.print(node.val);
            if (node.next != null) {
                System.out.print("->");
            }
            node = node.next;
        }
    }
  public static ListNode removeElements(ListNode head, int val) //函数核心代码位置
```

ACM 读入二叉树结构

```
import java.util.*;
class TreeNode
{
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x)
    {
        this.val = x;
    }
}
public class Tree_Node {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        String[] split = s.split(" ");
        int[] arr = new int[split.length];
        for(int i = 0; i < arr.length; i++){
            arr[i] = Integer.parseInt(split[i]);
        }
        List<TreeNode> list = new ArrayList<>();
        Collections.fill(list, null);
        TreeNode root = null;
        for(int i = 0; i < arr.length; i++){
            TreeNode node = null;
            if(arr[i] != -1){
                node = new TreeNode(arr[i]);
            }
            list.add(i,node);
            if(i == 0){
                root = node;//root就是生成的树
            }
        }
        for (int i = 0;  2 * i + 2 < arr.length ; i++) {
```

```
            if(list.get(i) != null){
                list.get(i).left = list.get(2 * i + 1);
                list.get(i).right = list.get(2 * i + 2);
            }
        }
    }
}
```

数组

704. 二分查找

力扣题目链接

给定一个 n 个元素有序的（升序）整型数组 nums 和一个目标值 target ，写一个函数搜索 nums 中的 target，如果目标值存在返回下标，否则返回 -1。

示例 1:

```
输入: nums = [-1,0,3,5,9,12], target = 9
输出: 4
解释: 9 出现在 nums 中并且下标为 4
```

示例 2:

```
输入: nums = [-1,0,3,5,9,12], target = 2
输出: -1
解释: 2 不存在 nums 中因此返回 -1
```

提示:

- 你可以假设 nums 中的所有元素是不重复的。
- n 将在 [1, 10000]之间。
- nums 的每个元素都将在 [-9999, 9999]之间。

注意点:

- 左闭右闭 while （left <= right）

59.螺旋矩阵II

力扣题目链接

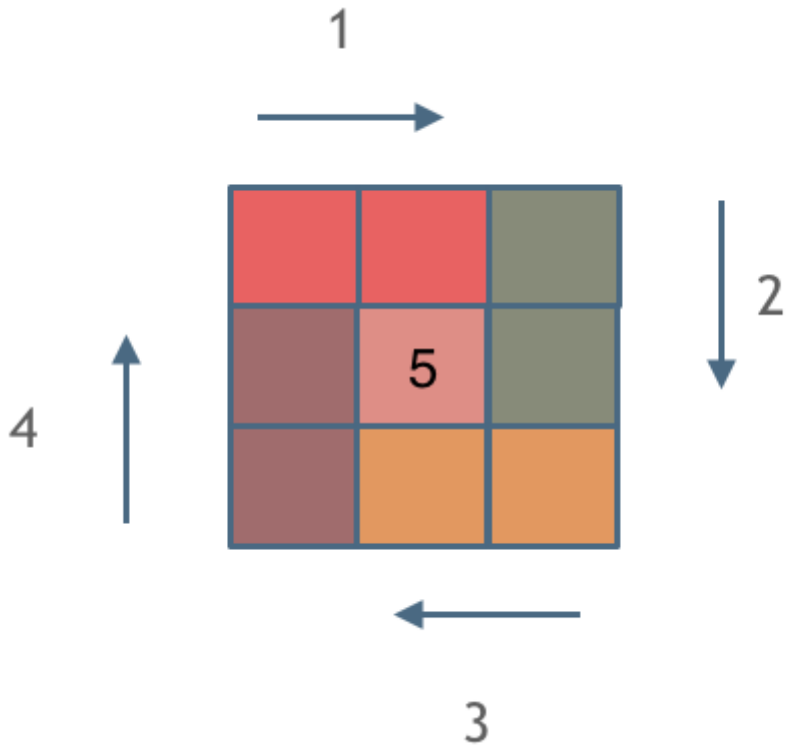
给定一个正整数 n，生成一个包含 1 到 n^2 所有元素，且元素按顺时针顺序螺旋排列的正方形矩阵。

示例:

输入: 3 输出: [[1, 2, 3], [8, 9, 4], [7, 6, 5]]

注意点:

- 奇数偶数 奇数的最里面那个元素 最后单独算
- 圈数为 n/2 例如4 就是2圈 3就是一圈 最后加个中间单独算的
- 左闭右开 通过while（圈数） 每圈分四个方向 左右 上下 右左 下上
- 每一圈结束后 下一圈的初始点 +1 ， 截止点-1 （这里可以借用圈数的变量来减）



3. 无重复字符的最长子串

难度 中等 7726 收藏 分享 切换为英文 接收动态 反馈

给定一个字符串 `s` ，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: s = "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: s = "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: s = "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。
```

注意点

- 用set来记录元素的存放 看看set里面是否存在这个元素，如果while 存在就从头开始删除直到不存在该数 就放进去
- 用res来记录最长的 res = Math.max(res,i-j+1);

双指针

第15题. 三数之和

力扣题目链接

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]]

注意点

- 首先需要排序Arrays.sort(nums)
- 然后i确定起始点 j，k是起始点后的数组的首尾指针，利用nums[i] + nums[j] + nums[k]和0的关系移动j k while来控制是否跳出循环
- [-1,-1,0,2] 这种例子可能会出现 [[-1,-1,2],[-1,-1,2]] 所以需要去重 我们回溯算法题中经常做 if(i > 0 && nums[i] == nums[i-1]){continue;}
- [-2,0,0,2,2] 这种例子就是也会重复 [[-2,0,2],[-2,0,2]] 就是当我们取到相等后 j k 下一个位置可能也会重复所以需要while来移动j k到不相等的位置
- 不用哈希表的原因就是去重很麻烦

堆栈

```
LinkedList
peekFirst()
peekLast()
pollFirst()
pollLast()
```

347.前 K 个高频元素

力扣题目链接

给定一个非空的整数数组，返回其中出现频率前 k 高的元素。

示例 1:

- 输入: nums = [1,1,1,2,2,3], k = 2
- 输出: [1,2]

示例 2:

- 输入: nums = [1], k = 1
- 输出: [1]

提示：

- 你可以假设给定的 k 总是合理的，且 1 ≤ k ≤ 数组中不相同的元素的个数。
- 你的算法的时间复杂度必须优于 $O(n \log n)$, n 是数组的大小。
- 题目数据保证答案唯一，换句话说，数组中前 k 个高频元素的集合是唯一的。
- 你可以按任意顺序返回答案。

注意点

```
map.put(num[i],map.getOrDefault(nums[i],0)+1)//map中常用，记录次数
PriorityQueue<Map.Entry<Integer, Integer>> queue = new PriorityQueue<>((o1, o2)
-> o2.getValue() - o1.getValue());//首先map不能直接存放转化成Entry的格式存放，按照指定的
比较器来存放 本题就利用优先队列的这一点 实现TopK o2-o1 由大到小
Set<Map.Entry<Integer, Integer>> set  = map.entrySet();//把哈希表转成entry形式存放的
列表
```

链表

142.环形链表II

力扣题目链接

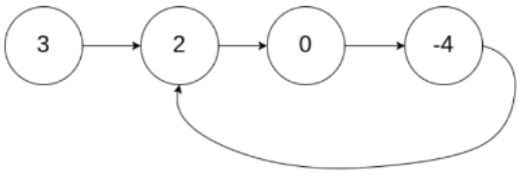
题意： 给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。

为了表示给定链表中的环，使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 pos 是 -1，则在该链表中没有环。

说明：不允许修改给定的链表。

示例 1：

```
输入：head = [3,2,0,-4], pos = 1
输出：tail connects to node index 1
解释：链表中有一个环，其尾部连接到第二个节点。
```



注意点

- 快慢指针 快走两步 慢走一步 如果相等代表有环
- 通过数学计算得到 **head到环入口距离==相遇时的快慢链表到环入口距离**
- 这样就可以找到环入口

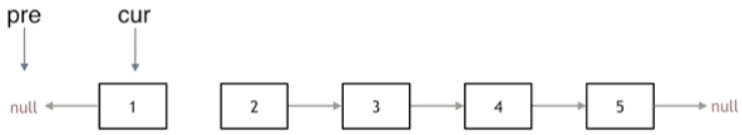
```
while(index1 != index2)
{
    index2  = index2 .next;
    index1  = index1 .next;
}
return index2;
```

206.反转链表

力扣题目链接

题意：反转一个单链表。

示例: 输入: 1->2->3->4->5->NULL 输出: 5->4->3->2->1->NULL



注意点

- 第一个点就是如上图 该如何反转
- 其次就在于 我必须定义一个tmp指针记录一下 cur.next指针　然后将cur指向pre也就是说断开

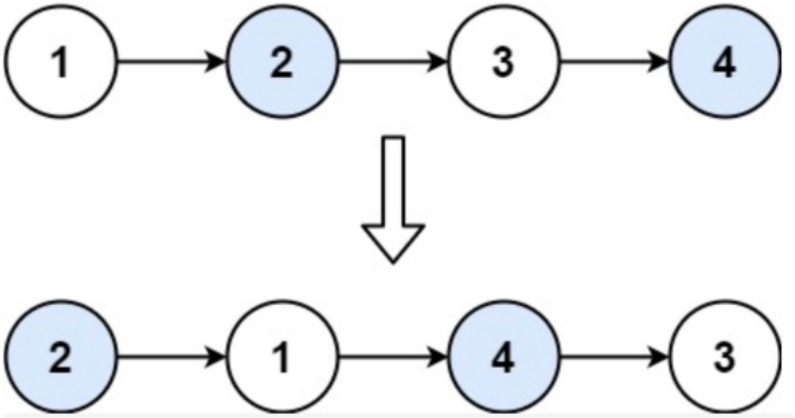
24. 两两交换链表中的节点

力扣题目链接

给定一个链表，两两交换其中相邻的节点，并返回交换后的链表。

你不能只是单纯的改变节点内部的值，而是需要实际的进行节点交换。

示例 1：



注意点

- 必须找到tmp 存放接下来的信息 不然断开连接后就没法继续存放了
- 然后就是画图 看看如何交换 再来就是确定下一次开始的位置 以及找到while截止的条件

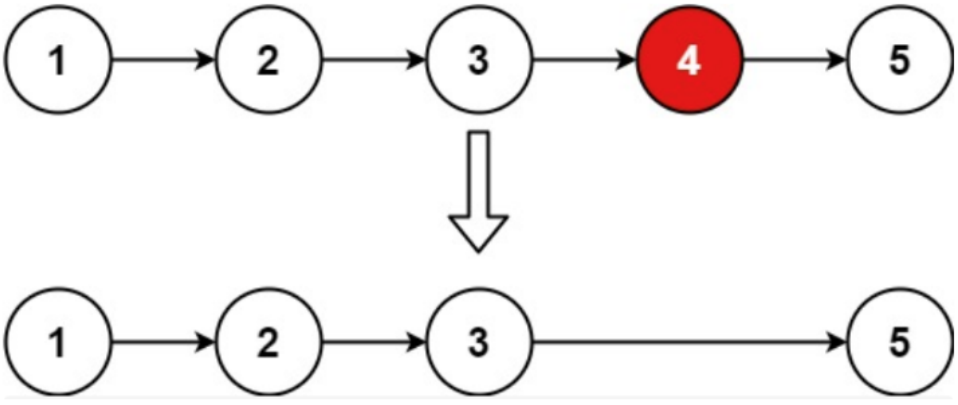
19.删除链表的倒数第N个节点

力扣题目链接

给你一个链表，删除链表的倒数第 n 个结点，并且返回链表的头结点。

进阶：你能尝试使用一趟扫描实现吗？

示例 1：



注意点

- 快慢指针 先让快指针先跑n个点 这里要让快指针先变成他的next 然后一起跑直到快指针null，这时候下一个节点就是要删除

二叉树

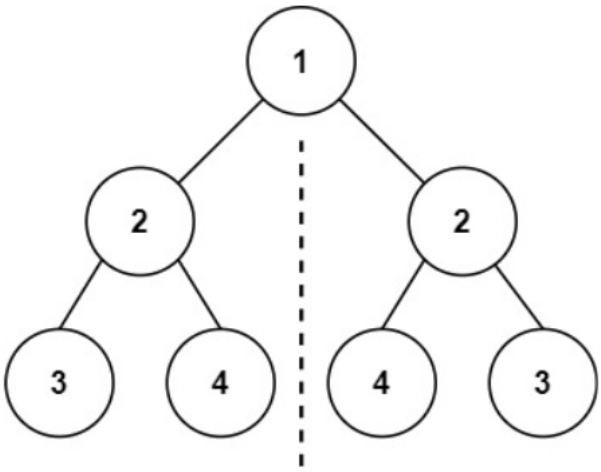
左右子树

101. 对称二叉树

难度 简单 1966 收藏 分享 切换为英文 接收动态 反馈

给你一个二叉树的根节点 `root` ， 检查它是否轴对称。

示例 1：



注意点

- 左子树为空右子树不为空为假
- 左子树不为空右子树为空为假
- 两个都不为空为真
- 左子树和右子树值不等为假
- 递归判断左子树的左子树和右子树的右子树
- 递归判断左子树的右子树和右子树的左子树
- 输出 **上面两个的与**
- 两个线程依次进行

617.合并二叉树

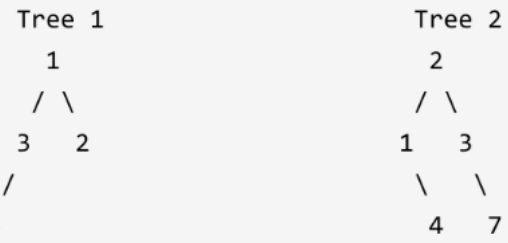
力扣题目链接

给定两个二叉树，想象当你将它们中的一个覆盖到另一个上时，两个二叉树的一些节点便会重叠。

你需要将他们合并为一个新的二叉树。合并的规则是如果两个节点重叠，那么将他们的值相加作为节点合并后的新值，否则不为 NULL 的节点将直接作为新二叉树的节点。

示例 1:

输入：



输出：

合并后的树：

注意点

- 左子树为空，输出右子树
- 右子树为空，输出左子树
- 利用前序遍历 先新建一棵树 把根节点设为两子树的val和
- 然后左子树递归
- 最后右子树递归

层序遍历

102.二叉树的层序遍历

力扣题目链接

给你一个二叉树，请你返回其按 层序遍历 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例：

二叉树：`[3,9,20,null,null,15,7]`，

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

返回其层次遍历结果：

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

注意点

- 定义两个容器 第一个容器是存放树结构的 第二个容器就是存放值的
- 通过长度记录需要加入的子节点长度

单调栈

739. 每日温度

难度 中等 1219 收藏 分享 切换为英文 接收动态 反馈

给定一个整数数组 `temperatures` ，表示每天的温度，返回一个数组 `answer` ，其中 `answer[i]` 是指对于第 `i` 天，下一个更高温度出现在几天后。如果气温在这之后都不会升高，请在该位置用 `0` 来代替。

示例 1:

```
输入: temperatures = [73,74,75,71,69,72,76,73]
输出: [1,1,4,2,1,1,0,0]
```

示例 2:

```
输入: temperatures = [30,40,50,60]
输出: [1,1,1,0]
```

示例 3:

```
输入: temperatures = [30,60,90]
输出: [1,1,0]
```

注意点

- 先把0放进去
- 判断栈不为空且大于栈顶元素
- 那么就找到对象了，栈顶元素这个索引位置的右侧最大值找到了就是即将放进去的索引
- 如果小于就放进去，元素都会放进去的，不会凭空消失
- 通过在栈中放索引

```
class Solution {
    LinkedList<Integer> list = new LinkedList<>();
    public int[] dailyTemperatures(int[] temperatures) {
        int[] res = new int[temperatures.length];
        list.add(0);
        for(int i = 1;i < temperatures.length;i++)
        {
            while(!list.isEmpty() && temperatures[i] >
temperatures[list.peekLast()])
            {
                res[list.peekLast()] = i - list.peekLast();
                list.pollLast();
            }
            list.add(i);
        }
        return res;
    }
}
```

496. 下一个更大元素 I

难度 简单 766 收藏 分享 切换为英文 接收动态 反馈

`nums1` 中数字 `x` 的 **下一个更大元素** 是指 `x` 在 `nums2` 中对应位置 **右侧** 的 **第一个** 比 `x` 大的元素。

给你两个 **没有重复元素** 的数组 `nums1` 和 `nums2` ，下标从 **0** 开始计数，其中 `nums1` 是 `nums2` 的子集。

对于每个 `0 <= i < nums1.length` ，找出满足 `nums1[i] == nums2[j]` 的下标 `j` ，并且在 `nums2` 确定 `nums2[j]` 的 **下一个更大元素** 。如果不存在下一个更大元素，那么本次查询的答案是 `-1` 。

返回一个长度为 `nums1.length` 的数组 `ans` 作为答案，满足 `ans[i]` 是如上所述的 **下一个更大元素** 。

示例 1：

```
输入：nums1 = [4,1,2], nums2 = [1,3,4,2].
输出：[-1,3,-1]
解释：nums1 中每个值的下一个更大元素如下所述：
- 4 ，用加粗斜体标识，nums2 = [1,3,4,2]。不存在下一个更大元素，所以答案是 -1 。
- 1 ，用加粗斜体标识，nums2 = [1,3,4,2]。下一个更大元素是 3 。
- 2 ，用加粗斜体标识，nums2 = [1,3,4,2]。不存在下一个更大元素，所以答案是 -1 。
```

示例 2：

```
输入：nums1 = [2,4], nums2 = [1,2,3,4].
输出：[3,-1]
解释：nums1 中每个值的下一个更大元素如下所述：
- 2 ，用加粗斜体标识，nums2 = [1,2,3,4]。下一个更大元素是 3 。
- 4 ，用加粗斜体标识，nums2 = [1,2,3,4]。不存在下一个更大元素，所以答案是 -1 。
```

- 首先将输出全赋值为-1
- 依旧对nums2进行单调栈的操作，当大于这个栈顶的索引出现的时候，判断栈顶的索引元素在nums1中出现没，出现就赋值

503. 下一个更大元素 II

难度 中等 657 收藏 分享 切换为英文 接收动态 反馈

给定一个循环数组 `nums` （ `nums[nums.length - 1]` 的下一个元素是 `nums[0]` ），返回 *`nums` 中每个元素的* ***下一个更大元素*** 。

数字 `x` 的 **下一个更大的元素** 是按数组遍历顺序，这个数字之后的第一个比它更大的数，这意味着你应该循环地搜索它的下一个更大的数。如果不存在，则输出 `-1` 。

示例 1:

```
输入: nums = [1,2,1]
输出: [2,-1,2]
解释: 第一个 1 的下一个更大的数是 2；
数字 2 找不到下一个更大的数；
第二个 1 的下一个最大的数需要循环搜索，结果也是 2。
```

示例 2:

```
输入: nums = [1,2,3,4,3]
输出: [2,3,4,-1,4]
```

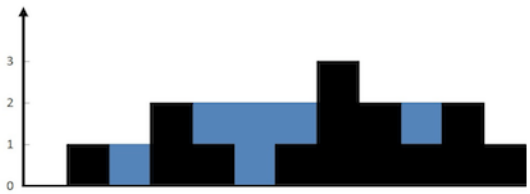
- 也全都设为-1
- 这题的意思就是可能循环找，最后一个元素也要找到全局的比他大的
- 这样就可以把nums复制两倍，然后通过取余的方式赋值进去，可以覆盖

42. 接雨水

难度 困难 3599 收藏 分享 切换为英文 接收动态 反馈

给定 `n` 个非负整数表示每个宽度为 `1` 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例 1:



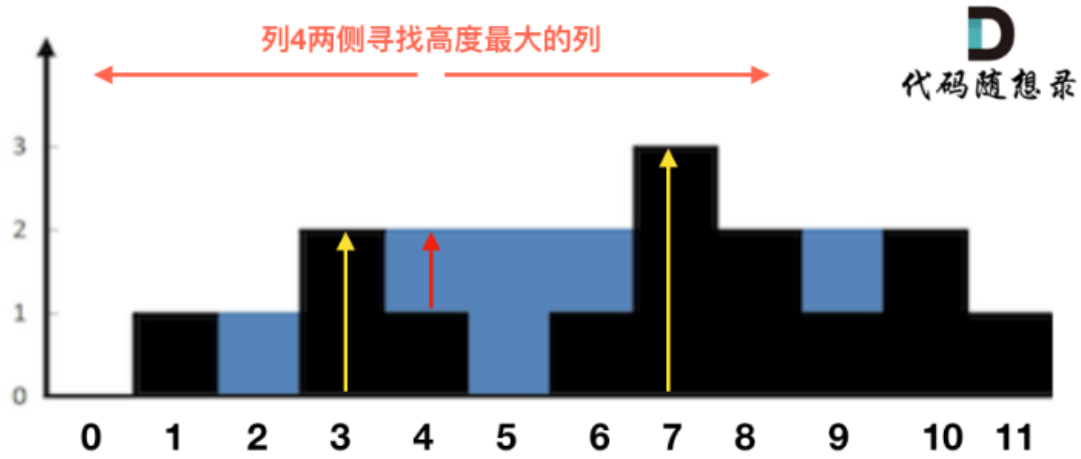

```
输入: height = [0,1,0,2,1,0,1,3,2,1,2,1]
输出: 6
解释: 上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。
```

示例 2:

```
输入: height = [4,2,0,3,2,5]
输出: 9
```

双指针法

- 从0遍历到最后
- 但是0和最后一个不算因为无法接雨滴 所以遇到这两个直接continue
- 本题重点找左右两个最高的位置这样才能接雨滴
- 先把当前默认的数组值，为这两个左右最高
- 然后从当前开始往左 往右找到右边最高的，和左边最高的，这样取两个中小的在减去自己就是我这一列能接的雨滴

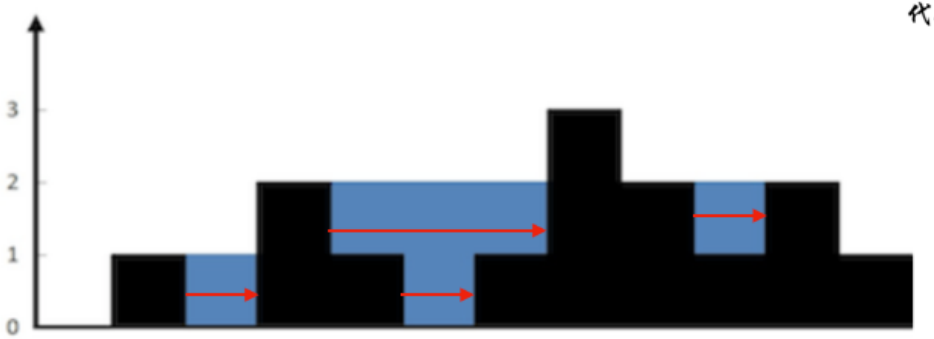


单调栈

- 三个元素确定可承载的雨滴，栈顶一个，栈顶下一个一个，即将进来的i一个
- 如果大于栈顶元素，那么栈顶元素polllast,此时栈里还有元素，那么一定大于刚才polllast的元素
- 所以直接求被polllast的两边的元素的最小值减去我本身得到我的高
- i-polllast左边那个元素 -1为宽
- 长×宽即可
- 按照行来算的

```
class Solution {
    public int trap(int[] height) {
        LinkedList<Integer> list = new LinkedList<>();
        int sum = 0;
        list.add(0);
        for(int i = 1;i < height.length;i++)
        {
            while(!list.isEmpty() && height[i] > height[list.peekLast()])
            {
                int temp = list.pollLast();
                if(!list.isEmpty())
                {
                    int high = Math.min(height[i],height[list.peekLast()]) -
height[temp];
                    int wideth = i - list.peekLast()-1;
                    sum += high*wideth;
                }
            }
            list.add(i);
        }
        return sum;
    }
}
```

按照行计算



上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。 **感谢 Marcos** 贡献此图。

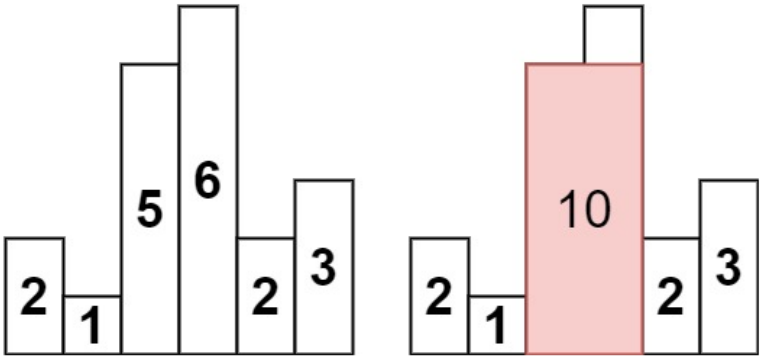
84. 柱状图中最大的矩形

难度 困难 2034 收藏 分享 切换为英文 接收动态 反馈

给定 *n* 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。

求在该柱状图中，能够勾勒出来的矩形的最大面积。

示例 1:




```
输入：heights = [2,1,5,6,2,3]
输出：10
解释：最大的矩形为图中红色区域，面积为 10
```

注意点

- 首尾加一个0
- while中找小于的
- 大于的话往里面放