题目1

小Q正在给一条长度为n的道路设计路灯安置方案。

为了让问题更简单,小Q把道路视为n个方格,需要照亮的地方用'.'表示, 不需要 照亮的障碍物格子用'X'表示。

小Q现在要在道路上设置一些路灯, 对于安置在pos位置的路灯, 这盏路灯可以照亮pos - 1, pos, pos + 1这三个位置。

小Q希望能安置尽量少的路灯照亮所有'.'区域, 希望你能帮他计算一下最少需 要多少盏路灯。

输入描述： 输入的第一行包含一个正整数t(1 <= t <= 1000), 表示测试用例数接下来每两行一个测试数据, 第一行一个正整数n(1 <= n <= 1000),表示道路的长度。第二行一个字符串s表示道路的构造,只包含'.'和'X'。

输出描述： 对于每个测试用例, 输出一个正整数表示最少需要多少盏路灯。

思路（贪心）：

1. 从位置i开始（假设位置i-1之前都已经被点了，且不会对位置i有影响）：
    1. 如果i位置为“X”，则i++；
    2. 如果i位置不是“X”，则路灯加一。如果i+1为“X”，则下次从i+2开始；如果i+1不是“X”，则下次从i+3开始；如果越界就break；

```
function lightNums(light) {
    const arr = light.split("");
    // 从i开始后续最少需要多少盏灯，0~i-1不会对i位置的安排产生印象；
    function process(arr, i) {
        if (i >= arr.length) {
            return 0;
        }
        let res = 0;
        if (arr[i] === "x") {
            res = process(arr, i+1);
        } else {
            if (arr[i+1] === ".") {
                res = process(arr, i+3) + 1;
            } else {
                res = process(arr, i+2) + 1;
            }
        }
        return res;
    }
    return process(arr, 0);
}
```

题目2

已知一棵二叉树中没有重复节点，并且给定了这棵树的中序遍历数组和先序遍历数组，返回后序遍历数组。

比如给定：

int[] pre = { 1, 2, 4, 5, 3, 6, 7 };

int[] in = { 4, 2, 5, 1, 6, 3, 7 };

返回：{ 4, 5, 2, 6, 7, 3, 1}

思路：

1. 定义递归process(pre, in, after, peri, perj, ini, inj, afteri, afterj)，其中pre、in、after为先序、中序、后序数组，peri, perj, ini, inj, afteri, afterj为数组中的范围。含义为用pre数组prei~prej范围和in数组ini~inj范围生成后序遍历after数组afteri~afterj范围，其中所有数组的范围等长；
2. pre的一个元素为after的最后一个元素，同时也是二叉树的头；根据头节点在in中的位置推出左树和右树的长度，调用process用pre和in的左树（右树）得出after的左树（右树）；

```
function getPosArr(pre, inArr) {
    const inMap = new Map();
    inArr.forEach((node, index) => {
        inMap.set(node, index)
    })
    const pos = new Array(pre.length);
    const process = (preStart, preEnd, inStart, inEnd, posStart, posEnd) => {
        if (posStart <= posEnd && posStart >= 0) {
            pos[posEnd] = pre[preStart];
        }
        let inMidIndex = inMap.get(pre[preStart]);
        let inLeftStart = inStart, inLeftEnd = inMidIndex - 1, inRightStart =
inMidIndex + 1, inRightEnd = inEnd;
        let preLeftStart = preStart + 1, preLeftEnd = preLeftStart + inLeftEnd -
inLeftStart, preRightStart = preLeftEnd + 1, preRightEnd = preEnd;
        let posLeftStart = posStart, posLeftEnd = posStart + preLeftEnd -
preLeftStart, posRightStart = posLeftEnd + 1, posRightEnd = posEnd - 1;

        if (preLeftStart <= preLeftEnd) {
            process(preLeftStart, preLeftEnd, inLeftStart, inLeftEnd,
posLeftStart, posLeftEnd)
        }
        if (preRightStart <= preRightEnd) {
            process(preRightStart, preRightEnd, inRightStart, inRightEnd,
posRightStart, posRightEnd)
        }
    }

    process(0, pre.length-1, 0, inArr.length-1, 0, pos.length-1)
    return pos;
}
```

题目三

把一个数字用中文表示出来。数字范围为 [0, 99999]。 为了方便输出，使用字母替换相应的中文，万W 千Q 百B 十S 零L。使用数字取代中 文数字注：对于 11 应该表示为 一十一(1S1)，而不是十一(S1) 输入描述： 数字

0（包含）到 99999（包含）。 输出描述： 用字母替换相应的中文，万W 千Q 百B 十S 零L 示例1: 输入 12001 输出 1W2QL1

思路：

从小到大依次解决，先解决0~9范围的打印num0_9，然后在是num0_99、num0_999、num0_9999。。。。

每次对数字取余数，调用前一个函数打印

题目四

求完全二叉树节点的个数

思路：

1. 获取二叉树的深度；
2. 判断右子树的最左节点所达到的最大深度；
    1. 如果最左节点到达的深度为整个二叉树的最大深度，代表左子树为满二叉树，节点个数为`2^h - 1`（h为左子树的高度）
    2. 如果最左节点到达的深度小于整个二叉树的最大深度，代表右子树为满二叉树；
3. 如果2中左子树（右子树）为满二叉树，则递归计算以右子树（左子树）为头的完全二叉树的节点个数；
4. 时间复杂度O((logn)^2)

```
function getCTNodeNum(head) {
    if (head === null) {
        return 0;
    }
    let h = getCTHeight(head);
    if (getCTHeight(head.right) + 1 === h) {
        // 左子树为满二叉树，高度为h-1
        return num = getCTNodeNum(head.right) + Math.pow(2, h - 1)

    } else {
        // 右子树为满二叉树，高度为h-2
        return num = getCTNodeNum(head.left) + Math.pow(2, h-2);
    }
}

function getCTHeight(head) {
    if (head === null) {
        return 0;
    }
    let height = 1;
    while (head.left) {
        height++;
        head = head.left;
    }
    return height;
}
```

题目五（动态规划-从左到右尝试）

最长递增子序列问题

如[3,1,2,6,3,4]，最长递增子序列为1，2，3，4

思路1：

1. 建立长度为arr.length的dp数组，dp[i]表示必须以 arr[i]结尾的情况下最长递增子序列的长度。
2. 假设1~i已经得出，dp[i+1]时需要找到1~i中比arr[i+1]小的数，判断得出dp[i+1];
3. 时间复杂度O(n^2);

```
var lengthOfLIS = function(nums) {
    const dp = new Array(nums.length).fill(1);
    let ans = 0;
    for (let i = 0; i < nums.length; i++) {
        for (let j = 0; j < i; j++) {
            if (nums[i] > nums[j]) {
                dp[i] = Math.max(dp[i], dp[j] + 1);
            }
        }
        ans = Math.max(ans, dp[i])
    }

    return ans;
};
```

思路2（优化）：

1. 建立长度为arr.length的dp数组，dp[i]表示必须以 arr[i]结尾的情况下最长递增子序列的长度。
2. 建立长度为arr.length的ends数组，ends[i]表示所有长度为i+1的递增子序列中最小的结尾（ends数组严格有序的）。
3. 计算dp[i+1]时，在ends二分查找大于arr[i+1]最左的位置，如果存在替换ends中的值，如果不存在则在ends添加arr[i+1]。dp[i+1]为ends下标+1。

```
var lengthOfLIS = function(nums) {
    // ends[i]表示所有长度为i+1的递增子序列中最小的结尾（ends数组严格有序的）。
    const ends = [Infinity];
    let ans = 0;
    for (let i = 0; i < nums.length; i++) {
        let left = 0;
        let right = ends.length - 1;
        while (left <= right) {
            let mid = ((right - left) >> 2) + left;
            if (nums[i] <= ends[mid]) {
                right = mid - 1;
            } else {
                left = mid + 1;
            }
        }
        if (right === ends.length - 1) {
            ends.push(nums[i]);
```

```
            } else {
                ends[left] = nums[i];
            }
            ans = Math.max(ans, ends.length)
        }
        return ans;
    };
```

题目六

小Q得到一个神奇的数列: 1, 12, 123,...12345678910,1234567891011...。

并且小Q对于能否被3整除这个性质很感兴趣。

小Q现在希望你能帮他计算一下从数列的第l个到第r个(包含端点)有多少个数可以被3整除。

输入描述:

输入包括两个整数l和r(1 <= l <= r <= 1e9), 表示要求解的区间两端。

输出描述:

输出一个整数, 表示区间内能被3整除的数字个数。

示例1:

输入

2 5

输出

3

解题思路：判断一个数能不能被3整除，等价于一个数的每位之和能否被3整除。刚开始想打表，但发现数据 量是1e9，一维数组最多只能开到1e8.所以就纯暴力判断了，不过数据是有规律的，第一个数是1、第二个数 是12，第三个数是123，所以只用判断n*(n+1)/2%3即可。因为数量太大了，所以用long long