482. License Key Formatting

Hint:
1.replace "-"
2.要判断新的字符串是否为空

```
class Solution {
    public String licenseKeyFormatting(String S, int K) {
        String newstring = S.replace("-","");
        if(newstring.equals("")){
            return "";
        }
        int length = newstring.length();
        List<Character> list = new ArrayList<>();
        for(int i = 1; i <= length; i++){
            list.add(0, newstring.charAt(length - i));
            if(i % K == 0){
                list.add(0, '-');
            }
        }
        if(list.get(0) == '-'){
            list.remove(0);
        }
        StringBuilder sb = new StringBuilder();
        for(int i = 0; i < list.size(); i++){
            sb.append(list.get(i));
        }
        return sb.toString().toUpperCase();
    }
}
```

113. Path Sum II

题目：返回路径和等于sum的路径list

Hint:
1.dfs
2.dfs的时候需要将上级list保存到新的list中

```
class Solution {
    List<List<Integer>> ans = new ArrayList<>();
    public List<List<Integer>> pathSum(TreeNode root, int sum) {
        List<Integer> list = new ArrayList<>();
        if(root == null){
            return ans;
        }
        dfs(root, sum, list);
        return ans;

    }

    private void dfs(TreeNode root, int sum, List<Integer> list){
        if(root.left == null && root.right == null && sum == root.val){
            list.add(root.val);
            ans.add(list);
        }
        if(root.left != null){
            List<Integer> leftlist = new ArrayList<>();
            leftlist.addAll(list);
            leftlist.add(root.val);
            dfs(root.left, sum - root.val, leftlist);
        }
        if(root.right != null){
            List<Integer> rightlist = new ArrayList<>();
            rightlist.addAll(list);
            rightlist.add(root.val);
            dfs(root.right, sum - root.val, rightlist);
        }
    }
}
```

129. Sum Root to Leaf Numbers

题目：求二叉树每一条从根到叶子的路径组成的数字组成的和

Hint:
1.dfs
2.用list存储到叶子时的路径组成的数字和

```
class Solution {
    public int sumNumbers(TreeNode root) {
        if(root == null){
            return 0;
        }
        List<Integer> list = new ArrayList<>();
        dfs(list, root, 0);
        int sum = 0;
        for(Integer temp: list){
            sum = sum + temp;
        }
        return sum;

    }

    private void dfs(List<Integer> list, TreeNode root, int sum){
        if(root.left == null && root.right == null){
            list.add(sum * 10 + root.val);
            return;
        }
        if(root.left != null){
            dfs(list, root.left, sum * 10 + root.val);
        }
        if(root.right != null){
            dfs(list, root.right, sum * 10 + root.val);
        }
    }
}
```

261. Graph Valid Tree

题目：对于给定的图，判断能否构成tree

Hint:
1.bfs
2.用hashmap存储邻接矩阵
3.用boolean array存储每个点是否被访问过
4.加入queue前要判断是否已经访问（因为是无向图）

```
class Solution {
    public boolean validTree(int n, int[][] edges) {
        boolean [] visited = new boolean[n];
        Map<Integer, List<Integer>> map = new HashMap<>();
        for(int i = 0; i < n; i++){
            map.put(i, new ArrayList<>());
        }
        for(int i = 0; i < edges.length; i++){
            map.get(edges[i][0]).add(edges[i][1]);
            map.get(edges[i][1]).add(edges[i][0]);
        }
        Queue<Integer> queue = new LinkedList<>();
        queue.add(0);
        while(!queue.isEmpty()){
            Integer temp = queue.poll();
            if(visited[temp] == true){
                return false;
            }
            else{
                visited[temp] = true;
            }
            for(Integer a: map.get(temp)){
                if(visited[a] == false){
                    queue.add(a);
                }
            }
        }
        for(int i = 0; i < n; i++){
            if(visited[i] == false){
                return false;
            }
        }
        return true;
    }
}
```

250. Count Univalue Subtrees

题目：找出和子树value一样的节点数量

Hint: dfs

```
class Solution {
    int count = 0;
    public int countUnivalSubtrees(TreeNode root) {
        if(root == null){
            return count;
        }
        dfs(root);
        return count;
    }

    private boolean dfs(TreeNode root){
        if(root == null){
            return true;
        }
        boolean left = dfs(root.left);
        boolean right = dfs(root.right);
        if(left && right){
            if(root.left != null && root.val != root.left.val){
                return false;
            }
            if(root.right != null && root.val != root.right.val){
                return false;
            }
            count++;
            return true;
        }
        return false;
    }
}
```