- ## 733. Flood Fill

题目：给定image，从特点的点开始辐射，若数值相同则更改，数值不同则不变
Hint：DFS

```
class Solution {
    public int[][] floodFill(int[][] image, int sr, int sc, int newColor) {
        if(image[sr][sc] == newColor){
            return image;
        }
        else{
            int nowColor = image[sr][sc];
            dfs(image, sr, sc, nowColor, newColor);
        }
        return image;
    }

    private void dfs(int[][] image, int sr, int sc, int nowColor, int newColor){
        if(sr < 0 || sc < 0 || sr >= image.length || sc >= image[0].length){
            return;
        }
        if(image[sr][sc] != nowColor){
            return;
        }
        else{
            image[sr][sc] = newColor;
            dfs(image, sr, sc + 1, nowColor, newColor);
            dfs(image, sr, sc - 1, nowColor, newColor);
            dfs(image, sr + 1, sc, nowColor, newColor);
            dfs(image, sr - 1, sc, nowColor, newColor);
        }
    }
}
```

- ## 332. Reconstruct Itinerary

```
class Solution {
    public List<String> findItinerary(String[][] tickets) {
        Map<String, PriorityQueue<String>> map = new HashMap<>();
        for (int i = 0; i < tickets.length; i++) {
            if (!map.containsKey(tickets[i][0])) {
                map.put(tickets[i][0], new PriorityQueue<>());
            }
            map.get(tickets[i][0]).offer(tickets[i][1]);
        }
        List<String> result = new ArrayList<>();
        dfs("JFK", result, map);
        Collections.reverse(result);
        return result;
    }
    private void dfs(String s, List<String> result, Map<String, PriorityQueue<String>> map) {
        while (map.containsKey(s) && !map.get(s).isEmpty()) {
            dfs(map.get(s).poll(), result, map);
        }
        result.add(s);
    }
}
```

- ## 297. Serialize and Deserialize Binary Tree

题目：将二叉树序列化成字符串，再将字符串反序列化成二叉树
Hint：
1.用arraylist代替linkedlist进行level traverse
2.全部加入list后，再通过判断list结尾是不是符号和null进行删除修改
3.list加入到stringbuilder中，注意需要加入中括号

1.用boolean标志位标志是否需要从queue中取node和标志左节点还是优节点
2.null的情况处理

```

public class Codec {
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        if(root == null){
            return "[]";
        }
        StringBuilder sb = new StringBuilder();
        List<TreeNode> queue = new ArrayList<>();
        List<String> ans = new ArrayList<>();
        queue.add(root);

        for(int i = 0; i < queue.size(); i++){
            TreeNode temp = queue.get(i);
            if(temp == null){
                ans.add("null");
                ans.add(",");
                continue;
            }
            ans.add(String.valueOf(temp.val));
            ans.add(",");
            queue.add(temp.left);
            queue.add(temp.right);
        }
        while(ans.get(ans.size() - 1).equals("null") || ans.get(ans.size() - 1).equals(",")){
            ans.remove(ans.size() - 1);
        }
        sb.append("[");
        for(int i = 0; i < ans.size(); i++){
            sb.append(ans.get(i));
        }
        sb.append("]");
        return sb.toString();

    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        if(data.equals("[]")){
            return null;
        }
        String newdata = data.substring(1, data.length() - 1);
        String[] dataaftersplit = newdata.split(",");
        TreeNode root = new TreeNode(Integer.parseInt(dataaftersplit[0]));
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        boolean needToGetNode = true;
        TreeNode temp = null;
        for(int i = 1; i < dataaftersplit.length; i++){
            if(needToGetNode){
                temp = queue.poll();
                if(!dataaftersplit[i].equals("null")){
                    temp.left = new TreeNode(Integer.parseInt(dataaftersplit[i]));
                    queue.add(temp.left);
                }

            }
            else{
                if(!dataaftersplit[i].equals("null")){
                    temp.right = new TreeNode(Integer.parseInt(dataaftersplit[i]));
                    queue.add(temp.right);
                }
            }
            needToGetNode = !needToGetNode;
        }
        return root;
    }
}
```

- ## 114. Flatten Binary Tree to Linked List

```
class Solution {
    public void flatten(TreeNode root) {
        dfs(root);

    }
    public TreeNode dfs(TreeNode root){
        if(root == null){
            return null;
        }
        if(root.left == null && root.right == null){
            return root;
        }
        if( root.right == null){
            root.right = root.left;
            root.left = null;
            return dfs(root.right);
        }
        if(root.left == null){
            return dfs(root.right);
        }
        TreeNode newleft = dfs(root.left);
        TreeNode newright = dfs(root.right);
        newleft.right = root.right;
        root.right = root.left;
        root.left = null;
        return newright;

    }
}
```