117. Populating Next Right Pointers in Each Node II

题目： 将二叉树中节点的next指针指向节点的右方指针
Hint:
1.bfs
2.循环到最后一次的时候，该节点指向null

```
class Solution {
    public Node connect(Node root) {
        if(root == null){
            return null;
        }
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        while(!queue.isEmpty()){
            int size = queue.size();
            for(int i = 0; i < size; i++){
                Node temp = queue.poll();
                if(i == size - 1){
                    temp.next = null;
                }
                else{
                    temp.next = queue.peek();
                }
                if(temp.left != null){
                    queue.add(temp.left);
                }
                if(temp.right != null){
                    queue.add(temp.right);
                }
            }
        }
        return root;
    }
}
```

78. Subsets

题目： 对于给定array， 求出该array的所有subsets
Hint:
1.dfs
2.每一次dfs创建新的list存储当前dfs的结果

```
class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        ans.add(new ArrayList<Integer>());
        for(int i = 0; i < nums.length; i++){
            List<Integer> list = new ArrayList<>();
            dfs(ans, list, nums, i);
        }
        return ans;

    }

    private void dfs(List<List<Integer>> ans, List<Integer> list, int[] nums, int pos){
        if(pos >= nums.length){
            return;
        }

            List<Integer> newlist = new ArrayList<>();
            newlist.addAll(list);
            newlist.add(nums[pos]);
            ans.add(newlist);
        for(int i = pos; i < nums.length; i++){
            dfs(ans, newlist, nums, i + 1);
        }
    }
}
```

22. Generate Parentheses

题目： 给定一个整数，生成相应数量正确的括号排列
Hint:
1.用left和right分别标记左右括号的数量
2.dfs
3.对于dfs给出相应的跳出条件

```
class Solution {
    public List<String> generateParenthesis(int n) {
        List<String> ans = new ArrayList<>();
        dfs("", n - 1, n, ans);
        return ans;
    }
    private void dfs(String nowstring, int left, int right, List<String> ans){
        if(right < left){
            return;
        }
        if(left == 0 && right == 0){
            ans.add(nowstring);
            return;
        }
        if(left > 0){
            String firststring = new String(nowstring);
            firststring = firststring + "(";
            dfs(firststring, left - 1, right, ans);
        }
        if(right > 0){
            String secondstring = new String(nowstring);
            secondstring = secondstring + ")";
            dfs(secondstring, left, right - 1, ans);
        }
    }
}
```

17. Letter Combinations of a Phone Number

题目： 对于给定字符串，根据其数字返回九宫格键盘所有可能字母组合
Hint:
1. Dfs
2.用string array记录键盘数字
3.

```
class Solution {
    public List<String> letterCombinations(String digits) {
        String[][] dict = {{""}, {"a", "b", "c"}, {"d", "e", "f"}, {"g", "h", "i"}, {"j", "k", "l"},
                           {"m", "n", "o"}, {"p", "q", "r", "s"}, {"t", "u", "v"}, {"w", "x", "y", "z"}};
        List<String> ans = new ArrayList<>();
        if(digits.equals("")){
            return ans;
        }
        dfs(dict, "", digits, 0, ans);
        return ans;
    }

    private void dfs(String[][] dict, String nowstring, String digits, int pos, List<String> ans ){
        if(pos >= digits.length()){
            ans.add(nowstring);
            return;
        }
        int digitnum = digits.charAt(pos) - '0';
        for(int i = 0; i < dict[digitnum].length; i++){
            String newstring = new String(nowstring);
            newstring = newstring + dict[digitnum][i];
            dfs(dict, newstring, digits, pos + 1, ans);
        }
    }
}
```

46. Permutations

题目： 给定指定array且其元素不重复， 返回其元素组成的所有组合
Hint:
1.dfs
2.用set检验重复元素

```
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        if(nums.length == 0){
            return ans;
        }
        List<Integer> list = new ArrayList<>();
        dfs(nums, list, ans);
        return ans;
    }

    private void dfs(int[] nums, List<Integer> list, List<List<Integer>> ans){
        if(list.size() == nums.length){
            ans.add(list);
            return;
        }
        Set<Integer> set = new HashSet<>();
        for(int i = 0; i < list.size(); i++){
            set.add(list.get(i));
        }
        for(int i = 0; i < nums.length; i++){
            if(!set.contains(nums[i])){
                List<Integer> newlist = new ArrayList<>();
                newlist.addAll(list);
                newlist.add(nums[i]);
                dfs(nums, newlist, ans);
            }
        }
    }
}
```

79. Word Search

题目： 对于给定的字符串和矩阵， 返回在矩阵中是否有联系的字符能够组成该字符串
Hint:
1.用visited array来记录点是否访问过（hashset不行！！！）
2.dfs
3.将当前点记录为true后要修改回false

HashSet存放对象时，hashset存放的是对象的内存地址，因此int[]不能直接用contains比较，需要overide hashset的equals和hashCode函数

```
import java.util.*;
public class TestHashset {
        public static void main(String argc[]) {
                Set<int[]> set = new HashSet<>();
                int[] temp = new int [] {0,1};
                set.add(temp);
                int[] q =  {0,1};
                System.out.println(set.contains(q));
        }
}
```

因此上题答案为false

```
class Solution {
    public boolean exist(char[][] board, String word) {
        if(board.length == 0){
            return false;
        }
        boolean[][] visited = new boolean[board.length][board[0].length];
        for(int i = 0; i < board.length; i++){
            for(int j = 0; j < board[0].length; j++){
                if(word.charAt(0) == board[i][j]){
                    if(dfs(visited, board, 0, word, i, j)){
                        return true;
                    }
                }
            }
        }
        return false;
    }

    private boolean dfs(boolean[][] visited ,char[][] board, int pos, String word,  int row, int col){
        if(pos == word.length()){
            return true;
        }
        if(!inbound(board, row, col) || board[row][col] != word.charAt(pos) || visited[row][col]){
            return false;
        }
        visited[row][col] = true;
        if(dfs(visited, board, pos + 1, word, row + 1, col) ||
           dfs(visited, board, pos + 1, word, row - 1, col) ||
           dfs(visited, board, pos + 1, word, row, col + 1) ||
           dfs(visited, board, pos + 1, word, row, col - 1)){
            return true;
        }
        visited[row][col] = false;
        return false;
    }

    private boolean inbound(char[][] board, int row, int col){
        return row >= 0 && col >= 0 && row < board.length && col < board[0].length;
    }
}
```