- ## 904. Fruit Into Baskets

题目：找array中某两个数的最长子串长度
Hint:
1.O(n^2)用set记录当前数字的数量

```
class Solution {
    public int totalFruit(int[] tree) {
        int maxlength = 0;
        for(int i = 0; i < tree.length; i++){
            Set<Integer> set = new HashSet<>();
            int length = 0;
            for(int j = i; j < tree.length; j++){
                if(set.size() < 2){
                    set.add(tree[j]);
                    length++;
                    continue;
                } else{
                    if(set.contains(tree[j])){
                        length++;
                    } else{
                        break;
                    }
                }
            }
            if(length > maxlength){
                maxlength = length;
            }
        }
        return maxlength;
    }
}
```

- ## 54. Spiral Matrix

题目：将二维array螺旋输出
Hint:
1.用dir array记录方向
2.同时用visited记录array的访问情况
3.若已经访问或出界就转换方向

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> ans = new ArrayList<>();
        if(matrix.length == 0){
            return ans;
        }
        boolean[][] visited = new boolean[matrix.length][matrix[0].length];
        int size = matrix.length * matrix[0].length;
        int row = 0;
        int col = 0;
        int dir[][] = new int[][]{{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        int i = 0;
        while(ans.size() < size){
            ans.add(matrix[row][col]);
            visited[row][col] = true;
            if(inbound(matrix, row + dir[i][0], col + dir[i][1]) && visited[row + dir[i][0]][col + dir[i][1]] == false){
                row = row + dir[i][0];
                col = col + dir[i][1];
                continue;
            } else{
                if(i <= 2){
                    i++;
                    row = row + dir[i][0];
                    col = col + dir[i][1];
                } else{
                    i = 0;
                    row = row + dir[i][0];
                    col = col + dir[i][1];
                }
            }
        }
        return ans;
    }

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

遍历和分治法
1.分治法需要返回值
2.遍历无返回

104. Maximum Depth of Binary Tree

- 遍历法：

```
class Solution {
    int maxdepth;
    public int maxDepth(TreeNode root) {
        maxdepth = 0;
        traverse(root, 1);
        return maxdepth;
    }

    private void traverse(TreeNode node, int depth){
        if(node == null){
            return;
        }
        if(depth > maxdepth){
            maxdepth = depth;
        }
        traverse(node.left, depth + 1);
        traverse(node.right, depth + 1);
    }
}
```

- 分治法

```
class Solution {
    public int maxDepth(TreeNode root) {
        if(root == null){
            return 0;
        }
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        if(left > right){
            return left + 1;
        } else{
            return right + 1;
        }
    }
}
```

- ## 77. Combinations

题目：给定数字和个数，返回所有的特定个数的组合
Hint:
1.backtracking

```
class Solution {
    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> list = new ArrayList<>();
        addNumber(n, k, 1, list, ans);
        return ans;
    }

    private void addNumber(int n, int k, int num, List<Integer> list, List<List<Integer>> ans){
        if(list.size() == k){
            ans.add(list);
            return;
        }
        for(int i = num; i <= n; i++){
            List<Integer> newlist = new ArrayList<>(list);
            newlist.add(i);
            addNumber(n, k, i + 1, newlist, ans);
        }
    }
}
```

- ## 47. Permutations II

题目：给定数组（数字有重复），返回所有不重复的顺序组合

```
class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ans = new ArrayList<>();
        Set<Integer> set = new HashSet<>();
        List<Integer> nowlist = new ArrayList<>();
        dfs(nums, set, ans, nowlist);
        return ans;
    }

    private void dfs(int[] nums, Set<Integer>nowset, List<List<Integer>> ans, List<Integer> nowlist){
        if(nowset.size() == nums.length){
            boolean addToList = true;
            ans.add(nowlist);
            return;
        }
        for(int i = 0; i < nums.length; i++){
            if(!nowset.contains(i)){
                if(i >= 1 && nums[i] == nums[i - 1] && !nowset.contains(i - 1)){
                    continue;
                }
                List<Integer> newlist = new ArrayList<>(nowlist);
                newlist.add(nums[i]);
                Set<Integer> newset = new HashSet<>(nowset);
                newset.add(i);
                dfs(nums, newset, ans, newlist);
            }
            else{
                continue;
            }
        }
    }
}
```