357. Count Numbers with Unique Digits

题目：给定10的次方数，计算在这个数字以内的各位数都不一样的数字个数
Hint:
1.用visited[]记录每个数字的访问情况
2.用num[]来作为变量记录数量
3.经典backtracking

```
class Solution {
    public int countNumbersWithUniqueDigits(int n) {
        int count = 1;
        for(int i = 0; i < n; i++){
            count = count * 10;
        }
        int[] num = new int[]{0};
        boolean[] visited = new boolean[10];
        dfs(0, 0, visited, count, num);
        return num[0];
        |
    }

    private void dfs( int pos, int nowstring, boolean[] visited, int count, int[] num){
        if(pos != 0 && nowstring == 0){
            return;
        }
        for(int i = 0; i < 10; i++){
            if(visited[i] == false){
                int newstring = 10 * nowstring + i;
                if(newstring < count){
                    visited[i] = true;
                    num[0] = num[0] + 1;
                    dfs(pos + 1, newstring, visited, count, num);
                    visited[i] = false;
                } else{
                    break;
                }
            }
        }
    }
}
```

60. Permutation Sequence

题目：给定一个数n，用1到n组成各位不相同的n位数，按照从小到大排列，再给出k输出第k位
Hint:
1.用list存储所有结果
2.backtracking

```
class Solution {
    public String getPermutation(int n, int k) {
        int min = 1;
        for(int i = 0; i < n - 1; i++){
            min = min * 10;
        }
        boolean[] visited = new boolean[n + 1];
        List<Integer> list = new ArrayList<>();
        dfs(list, 0, min, visited, n);
        return String.valueOf(list.get(k - 1));
    }

    private void dfs(List<Integer> list, int nownum, int min, boolean[] visited, int n){
        if(nownum >= min){
            list.add(nownum);
        }
        for(int i = 1; i <= n; i++){
            if(visited[i] == false){
                int newnum = nownum * 10 + i;
                visited[i] = true;
                dfs(list, newnum, min, visited, n);
                visited[i] = false;
            }
        }
    }
}
```

216. Combination Sum III

题目:
1.给定一个目标数字，以及数列的位数
2.在1-9里搜索递增的数列组成，数列的和为目标数字
Hint:
1.backtracking

```
class Solution {
    public List<List<Integer>> combinationSum3(int k, int n) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> list = new ArrayList<>();
        dfs(k, list, ans, n, 0, 1);
        return ans;
    }

    private void dfs(int k, List<Integer> nowlist, List<List<Integer>> ans, int target, int sum, int pos){
        if(target == sum && nowlist.size() == k){
            ans.add(new ArrayList<>(nowlist));
            return;
        }
        if(sum > target){
            return;
        }
        for(int i = pos; i < 10; i++){
            int newsum = sum + i;
            nowlist.add(i);
            dfs(k, nowlist, ans, target, newsum, i + 1);
            nowlist.remove(nowlist.size() - 1);
        }
    }
}
```

90. Subsets II

题目：找出给定array的所有subset（有重复）
Hint:
1.backtracking
2.出现重复的取第一位

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

    private void dfs(int[] nums, int pos, List<Integer> list, List<List<Integer>> ans){
        ans.add(new ArrayList<>(list));
        for(int i = pos; i < nums.length; i++){
            if(i > pos && nums[i] == nums[i - 1]){
                continue;
            }
            list.add(nums[i]);
            dfs(nums, i + 1, list, ans);
            list.remove(list.size() - 1);
        }
    }
}
```

55. Jump Game

题目：对于给定array，array的每一个数字n代表能够向前走n格或以内
Hint:
1.DP
2.贪心
DP:

```
class Solution {
    public boolean canJump(int[] nums) {
        boolean[] length = new boolean[nums.length];
        length[0] = true;
        for(int i = 0; i < nums.length; i++){
            if(length[i] == true){
                int num = i + 1;
                for(int j = 0; j < nums[i]; j++){
                    if(num < nums.length){
                        length[num] = true;
                        num++;
                    } else{
                        break;
                    }
                }
            }
        }
        return length[nums.length - 1];
    }
}
```

贪心:

```
class Solution {
    public boolean canJump(int[] nums) {
        int cur = 0;
        for(int i = 0; i < nums.length; i++){
            int now = i + nums[i];
            if(nums[i] == 0){
                if(cur <= i && i != nums.length - 1){
                    return false;
                }
            }
            if(now > cur){
                cur = now;
            }
        }
        return cur >= nums.length - 1 ;
    }
}
```