267. Palindrome Permutation II

题目:
1.给定一串字符串，判断是否能组成回文，若可以，返回回文的组合，若不行，返回空list

Hint:
1.通过hashmap记录各个字符的数量
2.先通过hashmap判断能否组成回文
3.backtracking返回所有组合

```
class Solution {
    public List<String> generatePalindromes(String s) {
        List<StringBuilder> ans = new ArrayList<>();
        Map<Character, Integer> map = new HashMap<>();
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            map.put(ch, map.getOrDefault(ch, 0) + 1);
        }
        if(!isValid(map)){
            return new ArrayList<>();
        }
        int count = 0;
        for(Character a: map.keySet()){
            int countnum = map.get(a);
            if(countnum >= 2){
                count++;
            }
        }
        StringBuilder nowstring = new StringBuilder();
        dfs(ans, nowstring, map, 0, count);
        List<String> res = new ArrayList<>();
        for(StringBuilder sb: ans){
            if(sb.length() == s.length()){
                res.add(sb.toString());
            }
        }
        return res;
    }

    private boolean isValid(Map<Character, Integer> map){
        int count = 0;
        for(Character ch: map.keySet()){
            if(map.get(ch) % 2 == 1){
                count++;
            }
        }
        return count <= 1;
    }
    private void dfs(List<StringBuilder> ans, StringBuilder nowstring, Map<Character, Integer> map, int pos, int count){
        if(count == 0){
            boolean noleft = false;
            for(Character a:map.keySet()){
                if(map.get(a) != 0){
                    noleft = true;
                    break;
                }
            }
            if(noleft){
                for(Character a: map.keySet()){
                    if(map.get(a) == 1){
                        StringBuilder sb = new StringBuilder(nowstring.toString());
                        sb.insert(pos, a);
                        ans.add(sb);
                    }
                }
            } else{
                ans.add(nowstring);
            }

            return;
        }
        else{
            for(Character a: map.keySet()){
                int countnum = map.get(a);
                if(countnum >= 2){
                    StringBuilder sb = new StringBuilder(nowstring.toString());
                    sb.insert(pos, a);
                    sb.insert(pos, a);
                    map.put(a, map.get(a) - 2);
                    int temp = count;
                    if(map.get(a) <= 1){
                        count--;
                    }
                    dfs(ans, sb, map, pos + 1, count);
                    map.put(a, map.get(a) + 2);
                    count = temp;
                }
            }
        }
    }
}
```

254. Factor Combinations

题目：给出一个数字，返回该数字所有因数的组合
Hint:
1.backtracking
2.因数递增，因此dfs时要从上一级的factor开始

```
class Solution {
    public List<List<Integer>> getFactors(int n) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> list = new ArrayList<>();
        dfs(n, list, ans, 2);
        return ans;
    }

    private void dfs(int n, List<Integer> list, List<List<Integer>> ans, int start){
        if(n == 1){
            if(list.size() > 1){
                ans.add(new ArrayList<>(list));
            }
            return;
        }
        for(int i = start; i <= n; i++){
            if(n % i == 0){
                list.add(i);
                int newn = n / i;
                dfs(newn, list, ans, i);
                list.remove(list.size() - 1);
            }
        }
    }
}
```

31. Next Permutation

题目:
1.给定一个array，变换位置，输出下一个比原array大的array，若不存在则输出最小一个
Hint:
1.从尾端找出递增的位置pos及其数值cur
2.从尾端找出比cur大的最小值 并与cur交换
3.用quick sort将pos后的array排序

```
class Solution {
    public void nextPermutation(int[] nums) {
        int first = -1;
        for(int i = nums.length - 1; i >= 1; i--){
            if(nums[i] > nums[i - 1]){
                first = i - 1;
                break;
            }
        }
        if(first == -1){
            Arrays.sort(nums);
        } else
        {
            sort(nums, first);
        }
    }

    private void sort(int[] nums, int pos){
        int min = Integer.MAX_VALUE;
        int minIndex = 0;
        for(int i = nums.length - 1; i > pos; i--){
            if(nums[i] < min && nums[i] > nums[pos]){
                min = nums[i];
                minIndex = i;
            }
        }
        nums[minIndex] = nums[pos];
        nums[pos] = min;
        sortTarget(nums, pos + 1, nums.length - 1);
    }
    private void sortTarget(int[] nums, int start, int end){
        if(start >= end){
            return;
        }
        int left = start;
        int right = end;
        int mid = (left + right) / 2;
        while(left <= right){
            while(left <= right && nums[left] < nums[mid]){
                left++;
            }
            while(left <= right && nums[right] > nums[mid]){
                right--;
            }
            if(left <= right){
                int temp = nums[left];
                nums[left] = nums[right];
                nums[right] = temp;
                right--;
                left++;
            }
            sortTarget(nums, start, right);
            sortTarget(nums, left, end);
        }
    }
}
```