给定**不同**数字的集合，返回所有可能的排列。

例如，  
[1,2,3]有以下排列：

[

[1,2,3]，

[1,3,2]，

[2,1,3]，

[2,3,1]

[3,1,2]

[3,2,1]

]

所给的数列不包含重复项

代码

public List<List<Integer>> permute(int[] nums) {

List<List<Integer>> list = new ArrayList<>();

// Arrays.sort(nums); // not necessary

backtrack(list, new ArrayList<>(), nums);

return list;

}

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums){

if(tempList.size() == nums.length){ //当单个结果长度够的时候添加

list.add(new ArrayList<>(tempList));

} else{ //不够的时候

for(int i = 0; i < nums.length; i++){

if(tempList.contains(nums[i])) continue; //当前元素已经被添加过了，重复，跳过

tempList.add(nums[i]);//加入结果

backtrack(list, tempList, nums);//递归添加

tempList.remove(tempList.size() - 1);//到3结束后回溯

}

}

}

**当所给序列带有重复数字的时候**

给定可能包含重复的数字的集合，返回所有可能的唯一排列。

例如，  
[1,1,2]有以下独特的排列：

[

[1,1,2]

[1,2,1]

[2,1,1]

]

public List<List<Integer>> permuteUnique(int[] nums) {

List<List<Integer>> list = new ArrayList<>();

Arrays.sort(nums);

backtrack(list, new ArrayList<>(), nums, new boolean[nums.length]);

return list;

}

//uesd保存每个数据元素是否使用过，初始都为false

private void backtrack(List<List<Integer>> list, List<Integer> tempList, int [] nums, boolean [] used){

if(tempList.size() == nums.length){

list.add(new ArrayList<>(tempList));

} else{

for(int i = 0; i < nums.length; i++){

if(used[i] || i > 0 && nums[i] == nums[i-1] && !used[i - 1]) continue;

//当前的i用过了，或者与i相同的i-1的数据用过了，都跳过

used[i] = true; //当前数据没用过，与之相同的数据也没用过，设置为用

tempList.add(nums[i]); //添加到结果中

backtrack(list, tempList, nums, used); //递归

used[i] = false; //回溯，设置该元素为没用过

tempList.remove(tempList.size() - 1);//删除该元素

}

}

}