# 题目

给定一个不含重复数字的数组 nums ，返回其 所有可能的全排列 。你可以 按任意顺序 返回答案。

示例 1：

输入：nums = [1,2,3]

输出：[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

示例 2：

输入：nums = [0,1]

输出：[[0,1],[1,0]]

示例 3：

输入：nums = [1]

输出：[[1]]

提示：

1 <= nums.length <= 6

-10 <= nums[i] <= 10

nums 中的所有整数 互不相同

# 分析

## 方法一：回溯法

思路：

回溯法通常用于求解所有的排列或组合问题。具体思路如下：

1、初始化一个空的结果集 result，一个空的当前排列 perm，和一个标记数组 used，用来标记每个数字是否被使用过。

2、编写一个递归函数 dfs，该函数的作用是将当前未使用的数字加入到当前排列中，并标记该数字已被使用，然后递归调用 dfs 继续添加下一个未使用的数字，直到当前排列的长度等于 nums 的长度，将当前排列加入到结果集中。

3、在主函数中，调用 dfs 函数进行递归，然后返回结果集。

代码：

class Solution {

public:

vector<vector<int>> permute(vector<int>& nums) {

vector<vector<int>> result;

vector<int> perm;

vector<bool> used(nums.size(), false); // 标记数组，用来标记每个数字是否被使用过

dfs(nums, used, perm, result);

return result;

}

private:

void dfs(vector<int>& nums, vector<bool>& used, vector<int>& perm, vector<vector<int>>& result) {

if (perm.size() == nums.size()) { // 当当前排列的长度等于 nums 的长度时，将当前排列加入结果集

result.push\_back(perm);

return;

}

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

if (!used[i]) { // 如果当前数字未被使用过

perm.push\_back(nums[i]); // 将当前数字加入当前排列

used[i] = true; // 标记当前数字已被使用

dfs(nums, used, perm, result); // 递归调用，继续添加下一个未使用的数字

perm.pop\_back(); // 回溯，将当前数字移出当前排列，继续下一轮循环

used[i] = false; // 标记当前数字未被使用

}

}

}

};

## 方法二：递归法

要使用递归法实现全排列，可以按照以下步骤进行：

1、定义递归函数permuteRecursive，参数包括当前要处理的数组nums，当前处理的索引start，用于存储每个排列的结果perm，以及最终的结果集result。

2、如果start等于nums的大小，表示当前排列已经完成，将其加入到结果集中。

3、否则，从start开始，依次将每个元素与start位置交换，并递归处理剩下的部分，然后再换回来，保持数组的原样，以便下一轮交换。

4、在主函数中，初始化result和perm，然后调用permuteRecursive函数。

5、最后返回结果集result。

代码：

class Solution {

public:

vector<vector<int>> permute(vector<int>& nums) {

vector<vector<int>> result;

vector<int> perm;

permuteRecursive(nums, 0, perm, result);

return result;

}

private:

void permuteRecursive(vector<int>& nums, int start, vector<int>& perm, vector<vector<int>>& result) {

if (start == nums.size()) {

result.push\_back(perm);

return;

}

for (int i = start; i < nums.size(); ++i) {

swap(nums[start], nums[i]);

perm.push\_back(nums[start]);

permuteRecursive(nums, start + 1, perm, result);

perm.pop\_back();

swap(nums[start], nums[i]);

}

}

};