# 回溯、DFS

## 0046 全排列

链接：<https://leetcode-cn.com/problems/permutations/>

标签：回溯

### 精选题解

* 回溯算法入门级详解 + 练习（持续更新）
  + <https://leetcode-cn.com/problems/permutations/solution/hui-su-suan-fa-python-dai-ma-java-dai-ma-by-liweiw/>
* 官方题解
  + <https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-leetcode-solution-2/>
* **※ C++ 回溯法/交换法/stl 简洁易懂的全排列**
  + <https://leetcode-cn.com/problems/permutations/solution/c-hui-su-fa-jiao-huan-fa-stl-jian-ji-yi-dong-by-sm/>
* 精选代码
  + <https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-leetcode-solution-2/532710/>

### 方法一：标记数组

| **0046 全排列.cpp** |
| --- |
| <https://leetcode-cn.com/submissions/detail/127476452/> |
| 1. /\* 2. \* 【46】C++ 回溯法/交换法/stl 简洁易懂的全排列 3. \* https://leetcode-cn.com/problems/permutations/solution/c-hui-su-fa-jiao-huan-fa-stl-jian-ji-yi-dong-by-sm/ 4. \*/ 5. class Solution { 6. public: 7. vector<vector<int>> res; 8. void backtrack(vector<int> &nums, vector<int> &current, vector<bool> &flags) { 9. **if** (current.size() == flags.size()) { 10. res.push\_back(current); 11. } **else** { 12. **for** (int i=0; i<nums.size(); ++i) { 13. **if** (not flags[i]) { // nums[i] not in current 14. current.push\_back(nums[i]); 15. flags[i] = true; 16. backtrack(nums, current, flags); 17. current.pop\_back(); 18. flags[i] = false; 19. } 20. } 21. } 22. } 23. vector<vector<int>> permute(vector<int>& nums) { 24. **if** (nums.empty()) { 25. **return** {}; 26. } 27. vector<bool> flags(nums.size(), false); // true: in current; false: not in current 28. vector<int> current; 29. backtrack(nums, current, flags); 30. **return** res; 31. } 32. }; |

### 方法二：交换元素

| **0046 全排列 - swap.cpp** |
| --- |
| https://leetcode-cn.com/submissions/detail/127482585/ |
| 1. /\* 2. \* 【46】C++ 回溯法/交换法/stl 简洁易懂的全排列 - 全排列 - 力扣（LeetCode） 3. \* https://leetcode-cn.com/problems/permutations/solution/c-hui-su-fa-jiao-huan-fa-stl-jian-ji-yi-dong-by-sm/ 4. \* 全排列 - 全排列 - 力扣（LeetCode） 5. \* https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-leetcode-solution-2/ 6. \*/ 7. class Solution { 8. public: 9. vector<vector<int>> res; 10. void backtrack(vector<int> &nums, int start, int end) { 11. **if** (start == end) { 12. res.push\_back(nums); 13. } **else** { 14. **for** (int i=start; i<=end; ++i) { 15. swap(nums[i], nums[start]); 16. backtrack(nums, start+1, end); 17. swap(nums[i], nums[start]); 18. } 19. } 20. } 21. vector<vector<int>> permute(vector<int>& nums) { 22. **if** (nums.empty()) { 23. **return** {}; 24. } **else** { 25. backtrack(nums, 0, nums.size()-1); 26. **return** res; 27. } 28. } 29. }; |

## 0047 全排列 II

题目：<https://leetcode-cn.com/problems/permutations-ii/>

标签：回溯

### 精选题解

* 官方题解
  + <https://leetcode-cn.com/problems/permutations-ii/solution/quan-pai-lie-ii-by-leetcode-solution/>

### 关键思路

定义一个标记数组visited来标记已经填过的数。若visited[i] 为true，表示第i个数已经使用了；若visited[i] 为false，表示第i个数尚未使用。

要解决重复问题，只需保证在填第i个数时，重复数字只被填入一次。方法：对原数组排序，保证相同数字都相邻，然后每次填入的数一定是这个数所在重复数集合中「从左往右第一个未被填过的数字」，即如下的判断条件：

1. if (i > 0 && nums[i] == nums[i-1] && !visited[i-1]) {
2. continue;
3. }

假如排完序后的完整数组nums中有三个连续的数，那么一定只有如下4种状态：[×, ×, ×]，[√, ×, ×]，[√, √, ×]，[√, √, √]。（√ 表示已在生成的数组中，× 表示未在生成的数组中。）

### 复杂度

详见官方题解。

* 时间复杂度：O(n\*n!)。回溯复杂度O(n!)；每次新的生成数组需要复制n个元素。
* 空间复杂度：SO(n)。长度为n的标记数组；递归时深度最大为n。

### 代码

| **0047 全排列 II.cpp** |
| --- |
| https://leetcode-cn.com/problems/permutations-ii/submissions/ |
| 1. /\* 2. \* 全排列 II - 全排列 II - 力扣（LeetCode） 3. \* https://leetcode-cn.com/problems/permutations-ii/solution/quan-pai-lie-ii-by-leetcode-solution/ 4. \*/ 5. class Solution { 6. vector<int> visited; 7. public: 8. void backtrack(vector<int> &nums, vector<vector<int>> &res, int idx, vector<int> &current) { 9. **if** (idx==nums.size()) { 10. res.emplace\_back(current); 11. // \* C++ STL vector 添加元素（push\_back()和emplace\_back()）详解 12. // \* http://c.biancheng.net/view/6826.html 13. // push\_back() 向容器尾部添加元素时，首先会创建这个元素，然后再将这个元素拷贝或者移动到容器中（如果是拷贝的话，事后会自行销毁先前创建的这个元素）； 14. // 而 emplace\_back() 在实现时，则是直接在容器尾部创建这个元素，省去了拷贝或移动元素的过程。 15. **return** ; 16. } 17. **for** (int i=0; i<nums.size(); ++i) { 18. // 哪些情况不取当前的元素： 19. // 1. 已经访问过/在当前路径数组中 20. // 2. 和前一个数相等，且前一个数未被填过（表明该数不是第一个未填的数，故仍然跳过） 21. // 反过来理解，如果前一个相等的数已经被填过，那么此时就可以插入这后一个相等的数了， 22. // 因为我们在上一层嵌套中，已经保证前一个数当时是第一个未被填过的数了 23. // 此时意味着我们在当前路径数组中存在多个相等的数了 24. **if** (visited[i] || (i>0 && nums[i]==nums[i-1] && !visited[i-1])) 25. **continue**; 26. current.emplace\_back(nums[i]); 27. visited[i] = true; 28. backtrack(nums, res, idx+1, current); 29. visited[i] = false; 30. current.pop\_back(); 31. } 32. } 33. vector<vector<int>> permuteUnique(vector<int>& nums) { 34. vector<vector<int>> res; 35. vector<int> current; 36. visited.resize(nums.size()); 37. sort(nums.begin(), nums.end()); 38. backtrack(nums, res, 0, current); 39. **return** res; 40. } 41. }; |