46. Permutations

Created	@July 19, 2020 11:06 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/permutations/
Importance	*****
Tag	Backtrack
Video	

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Example 2:

```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Example 3:

```
Input: nums = [1]
Output: [[1]]
```

Constraints:

- `1 <= nums.length <= 6`
- `10 <= nums[i] <= 10`

- All the integers of `nums` are **unique**.

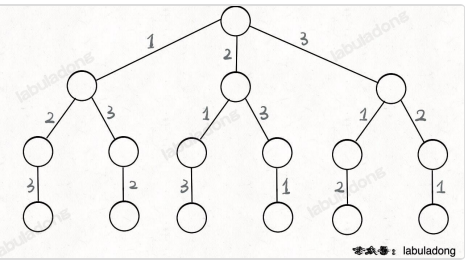
Solution

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        if not nums:
            return []

        results = []
        visited = [False] * len(nums)
        self.dfs(nums, [], visited, results)
        return results

    def dfs(self, nums, subset, visited, results):
        if len(nums) == len(subset):
            results.append(list(subset))
            return

        for i in range(len(nums)):
            if visited[i]:
                continue
            subset.append(nums[i])
            visited[i] = True
            self.dfs(nums, subset, visited, results)
            visited[i] = False
            subset.pop()
```

回溯算法解题套路框架

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 这篇文章是很久之前的一篇 回溯算法详解 的进阶版，之前

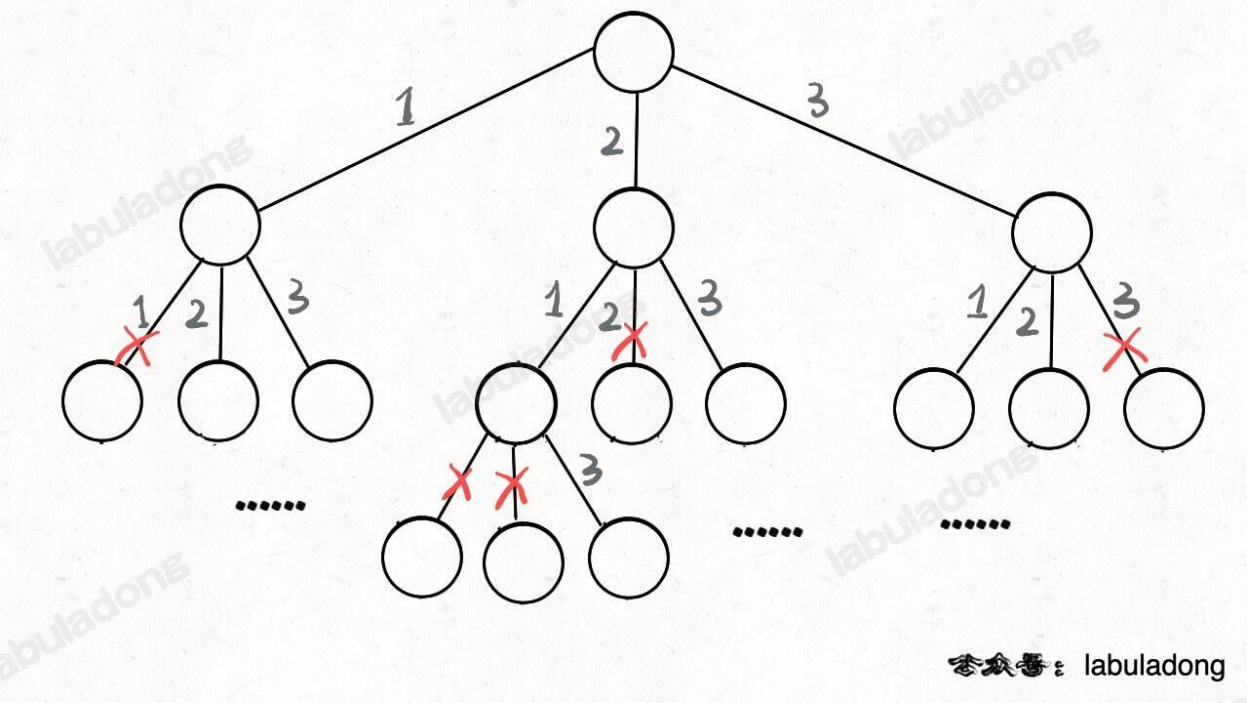
https://labuladong.github.io/algo/4/31/104/




```
List<List<Integer>> res = new LinkedList<>();

/* 主函数，输入一组不重复的数字，返回它们的全排列 */
List<List<Integer>> permute(int[] nums) {
    // 记录「路径」
    LinkedList<Integer> track = new LinkedList<>();
    // 「路径」中的元素会被标记为 true，避免重复使用
    boolean[] used = new boolean[nums.length];

    backtrack(nums, track, used);
```

```
    return res;
}

// 路径：记录在 track 中
// 选择列表：nums 中不存在于 track 的那些元素（used[i] 为 false）
// 结束条件：nums 中的元素全都在 track 中出现
void backtrack(int[] nums, LinkedList<Integer> track, boolean[] used) {
    // 触发结束条件
    if (track.size() == nums.length) {
        res.add(new LinkedList(track));
        return;
    }

    for (int i = 0; i < nums.length; i++) {
        // 排除不合法的选择
        if (used[i]) {
            // nums[i] 已经在 track 中，跳过
            continue;
        }
        // 做选择
        track.add(nums[i]);
        used[i] = true;
        // 进入下一层决策树
        backtrack(nums, track, used);
        // 取消选择
        track.removeLast();
        used[i] = false;
    }
}
```

why we need “used”?



但是必须说明的是，不管怎么优化，都符合回溯框架，而且时间复杂度都不可能低于 O(N!)，因为穷举整棵决策树是无法避免的。**这也是回溯算法的一个特点，不像动态规划存在重叠子问题可以优化，回溯算法就是纯暴力穷举，复杂度一般都很高**。

类似的题目：

- 51. N-Queens