0046. 全排列

ITCharge 大约 3 分钟

- 标签：数组、回溯
- 难度：中等

题目链接

- 0046. 全排列 - 力扣

题目大意

描述：给定一个不含重复数字的数组 `nums` 。

要求：返回其有可能的全排列。

说明：

- $1 \leq nums.length \leq 6$
- $-10 \leq nums[i] \leq 10$。
- `nums` 中的所有整数互不相同。

示例：

- 示例 1：

```py
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：

```py
输入：nums = [0,1]
输出：[[0,1],[1,0]]
```

解题思路

思路 1：回溯算法

根据回溯算法三步走，写出对应的回溯算法。

1. **明确所有选择**：全排列中每个位置上的元素都可以从剩余可选元素中选出，对此画出决策树，如下图所示。
    -
2. **明确终止条件**：
    - 当遍历到决策树的叶子节点时，就终止了。即当前路径搜索到末尾时，递归终止。
3. **将决策树和终止条件翻译成代码：**
    1. 定义回溯函数：
        - `backtracking(nums)`：函数的参数是 `nums`（可选数组列表），全局变量是 `res`（存放所有符合条件结果的集合数组）和 `path`（存放当前符合条件的结果）。
        - `backtracking(nums)`：函数代表的含义是：递归在 `nums` 中选择剩下的元素。
    2. 书写回溯函数主体（给出选择元素、递归搜索、撤销选择部分）。
        - 从当前正在考虑元素，到数组结束为止，枚举出所有可选的元素。对于每一个可选元素：
            - 约束条件：之前已经选择的元素不再重复选用，只能从剩余元素中选择。
            - 选择元素：将其添加到当前子集数组 `path` 中。
            - 递归搜索：在选择该元素的情况下，继续递归选择剩下元素。
            - 撤销选择：将该元素从当前结果数组 `path` 中移除。

```py
for i in range(len(nums)):            # 枚举可选元素列表
    if nums[i] not in path:           # 从当前路径中没有出现的数字中选择
        path.append(nums[i])          # 选择元素
        backtracking(nums)            # 递归搜索
        path.pop()                    # 撤销选择
```

3. 明确递归终止条件（给出递归终止条件，以及递归终止时的处理方法）。
    - 当遍历到决策树的叶子节点时，就终止了。也就是存放当前结果的数组 `path` 的长度等于给定数组 `nums` 的长度（即 `len(path) == len(nums)`）时，递归停止。

思路 1：代码

```py
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        res = []    # 存放所有符合条件结果的集合
        path = []   # 存放当前符合条件的结果
        def backtracking(nums):              # nums 为选择元素列表
            if len(path) == len(nums):       # 说明找到了一组符合条件的结果
                res.append(path[:])          # 将当前符合条件的结果放入集合中
                return

            for i in range(len(nums)):       # 枚举可选元素列表
                if nums[i] not in path:      # 从当前路径中没有出现的数字中选择
                    path.append(nums[i])     # 选择元素
                    backtracking(nums)       # 递归搜索
                    path.pop()               # 撤销选择

        backtracking(nums)
        return res
```

思路 1：复杂度分析

- **时间复杂度**：$O(n \times n!)$，其中 n 为数组 `nums` 的元素个数。
- **空间复杂度**：$O(n)$。