0090. 子集 II

ITCharge 大约 5 分钟

- 标签：位运算、数组、回溯
- 难度：中等

题目链接

- 0090. 子集 II - 力扣

题目大意

描述：给定一个整数数组 `nums` ，其中可能包含重复元素。

要求：返回该数组所有可能的子集（幂集）。

说明：

- 解集不能包含重复的子集。返回的 中，子集可以按任意顺序排列。
- $1 \leq nums.length \leq 10$。
- $-10 \leq nums[i] \leq 10$。

示例：

- 示例 1：

```py
输入：nums = [1,2,2]
输出：[[],[1],[1,2],[1,2,2],[2],[2,2]]
```

解题思路

思路 1：回溯算法

数组的每个元素都有两个选择：选与不选。

我们可以通过向当前子集数组中添加可选元素来表示选择该元素。也可以在当前递归结束之后，将之前添加的元素从当前子集数组中移除（也就是回溯）来表示不选择该元素。

因为数组中可能包含重复元素，所以我们可以先将数组排序，然后在回溯时，判断当前元素是否和上一个元素相同，如果相同，则直接跳过，从而去除重复元素。

回溯算法解决这道题的步骤如下：

- 先对数组 `nums` 进行排序。
- 从第 `0` 个位置开始，调用 `backtrack` 方法进行深度优先搜索。
- 将当前子集数组 `sub_set` 添加到答案数组 `sub_sets` 中。
- 然后从当前位置开始，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：
  - 如果当前元素与上一个元素相同，则跳过当前生成的子集。
  - 将可选元素添加到当前子集数组 `sub_set` 中。
  - 在选择该元素的情况下，继续递归考虑下一个元素。
  - 进行回溯，撤销选择该元素。即从当前子集数组 `sub_set` 中移除之前添加的元素。

思路 1：代码

```py
class Solution:
    def backtrack(self, nums, in     res, path):
        res.append(path[:])

        for i in range(index, len(nums)):
            if i > index and nums[i] == nums[i - 1]:
                continue
            path.append(nums[i])
            self.backtrack(nums, i + 1, res, path)
            path.pop()

    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        res, path = [], []
        self.backtrack(nums, 0, res, path)
        return res
```

思路 1：复杂度分析

- **时间复杂度**：$O(n \times 2^n)$，其中 n 指的是数组 `nums` 的元素个数，2^n 指的是所有状态数。每种状态需要 $O(n)$ 的时间来构造子集。
- **空间复杂度**：$O(n)$，每种状态下构造子集需要使用 $O(n)$ 的空间。

思路 2：二进制枚举

对于一个元素个数为 `n` 的集合 `nums` 来说，每一个位置上的元素都有选取和未选取两种状态。我们可以用数字 `1` 来表示选取该元素，用数字 `0` 来表示不选取该元素。

那么我们就可以用一个长度为 `n` 的二进制数来表示集合 `nums` 或者表示 `nums` 的子集。其中二进制的每一位数都对应了集合中某一个元素的选取状态。对于集合中第 `i` 个元素（`i` 从 `0` 开始编号）来说，二进制对应位置上的 `1` 代表该元素被选取，`0` 代表该元素未被选取。

举个例子来说明一下，比如长度为 `5` 的集合 `nums = {5, 4, 3, 2, 1}` ，我们可以用一个长度为 `5` 的二进制数来表示该集合。

比如二进制数 `11111` 就表示选取集合的第 `0` 位、第 `1` 位、第 `2` 位、第 `3` 位、第 `4` 位元素，也就是集合 `{5, 4, 3, 2, 1}` ，即集合 `nums` 本身。如下表所示：

集合 nums 对应位置（下标）	**4**	**3**	**2**	**1**	**0**
二进制数对应位数	1	1	1	1	1
对应选取状态	选取	选取	选取	选取	选取

再比如二进制数 `10101` 就表示选取集合的第 `0` 位、第 `2` 位、第 `5` 位元素，也就是集合 `{5, 3, 1}` 。如下表所示：

集合 nums 对应位置（下标）	**4**	**3**	**2**	**1**	**0**
二进制数对应位数	1	0	1	0	1
对应选取状态	选取	未选取	选取	未选取	选取

再比如二进制数 `01001` 就表示选取集合的第 `0` 位、第 `3` 位元素，也就是集合 `{5, 2}` 。如下标所示：

集合 nums 对应位置（下标）	4	3	2	1	0
二进制数对应位数	0	1	0	0	1
对应选取状态	未选取	选取	未选取	未选取	选取

通过上面的例子我们可以得到启发：对于长度为 `5` 的集合 `nums` 来说，我们只需要从 `00000 ~ 11111` 枚举一次（对应十进制为 $0 \sim 2^4 - 1$）即可得到长度为 `5` 的集合 `S` 的所有子集。

我们将上面的例子拓展到长度为 `n` 的集合 `nums` 。可以总结为：

- 对于长度为 `5` 的集合 `nums` 来说，只需要枚举 $0 \sim 2^n - 1$（共 2^n 种情况），即可得到所有的子集。

因为数组中可能包含重复元素，所以我们可以先对数组进行排序。然后在枚举过程中，如果发现当前元素和上一个元素相同，则直接跳过当前生层的子集，从而去除重复元素。

思路 2：代码

```py
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        n = len(nums)                           # n 为集合 nums 的元素个数
        sub_sets = []                           # sub_sets 用于保存所有子集
        for i in range(1 << n):                 # 枚举 0 ~ 2^n - 1
            sub_set = []                        # sub_set 用于保存当前子集
            flag = True                         # flag 用于判断重复元素
            for j in range(n):  # 枚举第 i 位元素
                if i >> j & 1:  # 如果第 i 为元素对应二进制位为 1，则表示选取该元素
                    if j > 0 and (i >> (j - 1) & 1) == 0 and nums[j] == nums[j -
1]:
                        flag = False        # 如果出现重复元素，则跳过当前生成的子
集
                        break
```

```
                sub_set.append(nums[j]) # 将选取的元素加入到子集 sub_set 中
            if flag:
                sub_sets.append(sub_set)    # 将子集 sub_set 加入到所有子集数组
sub_sets 中
        return sub_sets                     # 返回所有子集
```

思路 2：复杂度分析

- **时间复杂度**：$O(n \times 2^n)$，其中 n 指的是数组 `nums` 的元素个数，2^n 指的是所有状态数。每种状态需要 $O(n)$ 的时间来构造子集。
- **空间复杂度**：$O(n)$，每种状态下构造子集需要使用 $O(n)$ 的空间。