90. Subsets II

Created	@April 5, 2021 1:59 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/subsets-ii/
Importance	
Tag	Backtrack Recursion
Video	https://www.youtube.com/watch?v=rtFHxiQAICA&list=PLH8TFsY0qnE2R9kf_9vahNY6pG9601z_4&index=59

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

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

Example 2:

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

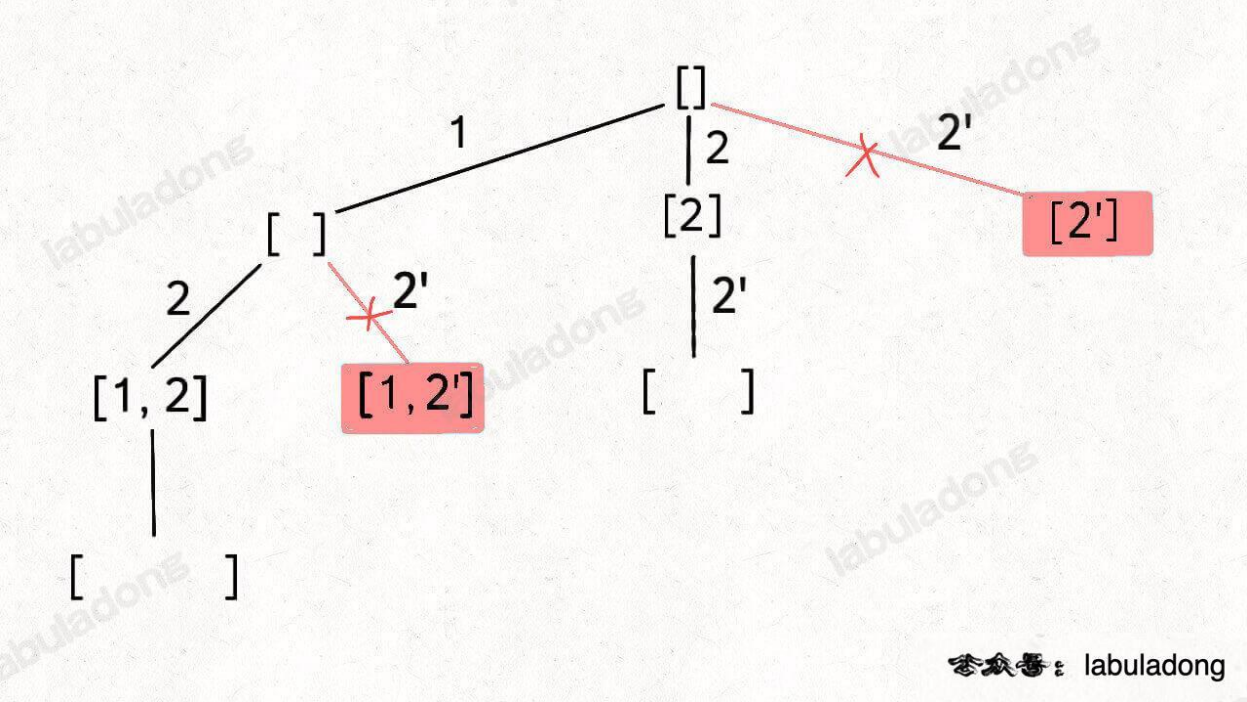
Constraints:

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

Solution

best solution;

所以我们需要进行剪枝，如果一个节点有多条值相同的树枝相邻，则只遍历第一条，剩下的都剪掉，不要去遍历：



体现在代码上，需要先进行排序，让相同的元素靠在一起，如果发现 `nums[i] == nums[i-1]`，则跳过

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

        nums.sort()
        self.dfs(nums, 0, [], results)
        return results

    def dfs(self, nums, startIndex, subset, results):
        results.append(list(subset))
        for i in range(startIndex, len(nums)):
            if i > startIndex and nums[i] == nums[i - 1]:
                continue
            subset.append(nums[i])
            self.dfs(nums, i + 1, subset, results)
            subset.pop()
```

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

public List<List<Integer>> subsetsWithDup(int[] nums) {
    // 先排序，让相同的元素靠在一起
    Arrays.sort(nums);
    backtrack(nums, 0);
    return res;
}

void backtrack(int[] nums, int start) {
    // 前序位置，每个节点的值都是一个子集
    res.add(new LinkedList<>(track));

    for (int i = start; i < nums.length; i++) {
        // 剪枝逻辑，值相同的相邻树枝，只遍历第一条
        if (i > start && nums[i] == nums[i - 1]) {
            continue;
        }
        track.addLast(nums[i]);
        backtrack(nums, i + 1);
        track.removeLast();
    }
}
```

better than the second one

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        results = []
        if not nums or len(nums) == 0:
            return results

        nums.sort()
        visited = [False] * len(nums)
        self.dfs(nums, 0, [], results, visited)
        return results

    def dfs(self, nums, startIndex, subset, results, visited):
        results.append(list(subset))
        for i in range(startIndex, len(nums)):
            if i != 0 and nums[i] == nums[i - 1] and visited[i - 1] == False:
                continue
            subset.append(nums[i])
            visited[i] = True
            self.dfs(nums, i + 1, subset, results, visited)
            visited[i] = False
            subset.pop()
```

Ref: 16 九章算法班2020版 subsets-ii_1

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        results = []
        if not nums or len(nums) == 0:
            return results

        nums.sort()
        self.dfs(nums, 0, [], results)
        return results

    def dfs(self, nums, startIndex, subset, results):
        results.append(list(subset))
        for i in range(startIndex, len(nums)):
            if i != 0 and nums[i] == nums[i - 1] and i > startIndex:
                continue
            subset.append(nums[i])
            self.dfs(nums, i + 1, subset, results)
            subset.pop()
```