1. 题目
2. 题干

给定一组**不含重复元素**的整数数组 nums，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。

1. 示例

输入: nums = [1,2,3]

输出:

[

[3],

  [1],

  [2],

  [1,2,3],

  [1,3],

  [2,3],

  [1,2],

  []

]

1. 题解
2. 思路

看到题目，脑子里出现的一定是递归+回溯，但是这不同于之前的递归+回溯，之前的递归解决的问题是寻找固定长度的数组，然而本题是寻找所有可能的子集，即不同长度的数组。如果不是固定长度的数组，那么递归退出的条件是什么呢？这里，我们依然使用数组的长度作为递归退出的条件，设定一个cur，当cur==nums.length时，就退出递归。这个cur代表着我们扫描一遍数组，作出nums.length次选择，如果选择该数就加入子集中，直到数组的结尾。cur的具体含义是选择的次数。

进入下一个递归的条件是只要当前的cur不等于nums.length，说明没有到数组的末尾，就将cur位置的数加入t子集数组中，且进入下一个递归dfs(nums, cur+1)。

回溯的操作从判断最后一个递归cur==nums.length后开始的，最后一个递归已经形成了一个子集，为了回到上一个递归的状态，从t子集数组中将最后一个数去除，回到上一个递归的状态，cur的值也相应地减掉1，那么，要形成另一个子集，就要cur+1，继续下一个递归dfs(nums, cur+1)，直到到达最终的递归返回位置cur==nums.length。这就是为什么多了一个dfs(nums, cur+1)的原因。

1. 代码实现

Java：

class Solution {

    List<Integer> t = new ArrayList<Integer>();

    List<List<Integer>> ans = new ArrayList<List<Integer>>();

    public List<List<Integer>> subsets(int[] nums) {

        dfs(nums, 0);

        return ans;

    }

    public void dfs(int[] nums, int cur){

        if(cur == nums.length){

            ans.add(new ArrayList<Integer>(t));

            return;

        }

        t.add(nums[cur]);

        dfs(nums, cur+1);

        t.remove(t.size()-1);

        dfs(nums, cur+1);

    }

}

C++：

class Solution {

public:

    vector<int> t;

    vector<vector<int>> ans;

    vector<vector<int>> subsets(vector<int>& nums) {

        dfs(nums, 0);

        return ans;

    }

    void dfs(vector<int>& nums, int cur){

        if(cur == nums.size()){

            ans.push\_back(t);

            return;

        }

        t.push\_back(nums[cur]);

        dfs(nums, cur+1);

        t.pop\_back();

        dfs(nums, cur+1);

    }

};