https://leetcode.cn/problems/subsets/

78. 子集

难度 中等 1965

给你一个整数数组 `nums` ，数组中的元素 **互不相同** 。返回该数组所有可能的子集（幂集）。

解集 **不能** 包含重复的子集。你可以按 **任意顺序** 返回解集。

示例 1：

```
输入：nums = [1,2,3]
输出：[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

示例 2：

```
输入：nums = [0]
输出：[[],[0]]
```

提示：

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- `nums` 中的所有元素 **互不相同**

通过次数 597,181 | 提交次数 736,614

求一个集合的所有子集，已知一个集合的非空子集个数是 2**n - 1

集合 {1, 2, 3, 4} 的非空子集分别为

{1} 1位

{1, 2} {1, 3} {1, 4} 2位

{1, 2, 3} {1, 2, 4} {1, 3, 4} 3位

{1, 2, 3, 4} 4位

{2} 1位

{2, 3} {2, 4} 2位

{2, 3, 4} 3位

{3} 1位

{3, 4} 2位

{4} 1位

2**4 - 1 = 15个

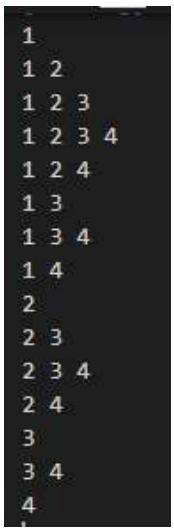
现使用程序来求出数组所有子集，并列出，不论顺序

由子集排列可知，我们可以按照**元素个数从少到多**、**元素大小从小到大**的方式进行**排列**

则有很通俗的方式，**全排列**，由下列程序易看出，想要求得元素个数最大为 n 时的所有子集，则需要进行 n 阶循环

上面讲的思想其实是**广度优先**的思想，实现起来麻烦一点，因为需要保存**当前元素个数为 n 的所有子集**的状态

由于使用的是深度优先搜索，所以 dfs 程序的输出如下。



```cpp
#include<bits/stdc++.h>
using namespace std;

/**
 *
 * vector<typename>:    泛型数组
 * push_back()          在尾端追加一个元素
 * pop_back()           移除尾端的一个元素
 *
 **/
int main()
    vector<int> arr = {1, 2, 3, 4};
    vector<vector<int>> res;
    vector<int> nums;
    // 当子集元素个数最大为1时
    /*
    for(int i = 0; i < 4; i++){
        nums.push_back(arr[i]);
        res.push_back(nums);
        nums.pop_back();
    }
    */
    // 当子集元素个数最大为2时
    /*
    for(int i = 0; i < 4; i++){
        nums.push_back(arr[i]);
        res.push_back(nums);
        for(int j = i+1; j < 4; j++){
            nums.push_back(nums[j]);
            res.push_back(nums);
            nums.pop_back();
        }
        nums.pop_back();
    }
    */
    // 当子集元素个数最大为3时
    for(int i = 0; i < 4; i++){
        nums.push_back(arr[i]);
        res.push_back(nums);
        for(int j = i+1; j < 4; j++){
            nums.push_back(nums[j]);
            res.push_back(nums);
            for(int k = j+1; k < 4; k++){
                nums.push_back(nums[k]);
                res.push_back(nums);
                nums.pop_back();
            }
            nums.pop_back();
        }
        nums.pop_back();
    }
    for(auto vec : res){
        for(auto num : vec){
            cout<<num<<' ';
        }
        cout<<endl;
    }
    return 0
}
```

对于上述程序，每多一个元素，就多一层循环，由此，我们可以利用调用方程的方式来代替子循环的实现，即递归

```cpp
#include<bits/stdc++.h>
using namespace std;

void dfs(vector<vector<int>> &res, vector<int> &nums, vector<int> &arr, int n){
    for(int i = n; i < arr.size(); i++){
        nums.push_back(arr[i]);
        res.push_back(nums);
        dfs(res, nums, arr, i+1);
        nums.pop_back();
    }
}

int main(){
    vector<int> arr = {1, 2, 3, 4};
    vector<vector<int>> res;
    vector<int> nums;
    dfs(res, nums, arr, 0);

    for(auto vec : res){
        for(auto num : vec){
            cout<<num<<' ';
        }
        cout<<endl;
    }
    return 0;
}
```

提交的代码： 1 年前

语言： cpp

```cpp
class Solution {
public:
    vector<vector<int>> res;
    vector<int> ans;

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

        for(int i = k; i < nums.size(); i++){
            ans.emplace_back(nums[i]);
            res.emplace_back(ans);
            dfs(nums, i+1);
            ans.pop_back();
        }
    }

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

        res.emplace_back(ans);
        dfs(nums, 0);
        return res;
    }
};
```

https://leetcode.cn/problems/the-number-of-beautiful-subsets/

2597. 美丽子集的数目

难度 中等 18

给你一个由正整数组成的数组 `nums` 和一个 **正** 整数 `k` 。

如果 `nums` 的子集中，任意两个整数的绝对差均不等于 `k` ，则认为该子数组是一个 **美丽** 子集。

返回数组 `nums` 中 **非空** 且 **美丽** 的子集数目。

`nums` 的子集定义为：可以经由 `nums` 删除某些元素（也可能不删除）得到的一个数组。只有在删除元素时选择的索引不同的情况下，两个子集才会被视作是不同的子集。

示例 1：

```
输入：nums = [2,4,6], k = 2
输出：4
解释：数组 nums 中的美丽子集有：[2], [4], [6], [2, 6] 。
可以证明数组 [2,4,6] 中只存在 4 个美丽子集。
```

示例 2：

```
输入：nums = [1], k = 1
输出：1
解释：数组 nums 中的美丽数组有：[1] 。
可以证明数组 [1] 中只存在 1 个美丽子集。
```

提示：

- `1 <= nums.length <= 20`
- `1 <= nums[i], k <= 1000`

通过次数 4,775 提交次数 14,373