78. 子集

入选理由

1. 昨天和大家打过预防针了，今天就是一个位运算应用题。 建议和我讲义里面的那道状态压缩一起食用。

标签

- 回溯
- 位运算

难度

- 中等

题目地址（78. 子集）

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

题目描述

```
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
```

前置知识

- 位运算
- 回溯

分析

这道题第一眼是可以用搜索/回溯来做的，每进行一次搜索就把当前结果存入结果集。这种求子集的类型题其实还有另一种做法：

每个元素有两种状态，拿或者不拿，那么如果一共有N个数，那就一共有2^N中可能，也就是有这么多个子集（子集包括全集和空集）。既然每一个数只有两种状态，那么我们不妨用一个 bit 来表示。这样题中的[1,2,3]，我们可以看成一个三个比特的组合：

比如 0 0 0 就代表空集，1 1 1 就代表全集， 1 0 0 就代表[1] (可正可反)。这样我们就可以进行位操作，$0 - 2^n - 1$的数的二进制数位为 1 的位置，就把对应的元素填入集合中。

PS: ((1 << i)& sign) != 0 的意思是用第 i 位是 1 比特与当前 sign 相与，若结果不为 0 就代表第 i 位比是 1

进阶: 用回溯解法解决该问题

代码

代码支持 Java，Python,CPP,JS

Java Code:

```
class Solution {

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

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

        int start = 0, end = 1 << nums.length;

        for (int sign = start; sign < end; sign++) {

            List<Integer> list = new LinkedList<>();

            for (int i = 0; i < nums.length; i++)
                if (((1 << i) & sign) != 0)
                    list.add(nums[i]);

            res.add(list);
```

```
        }

        return res;
    }
}
```

Python Code:

```
class Solution:
    def subsets(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        res, end = [], 1 << len(nums)
        for sign in range(end):
            subset = []
            for i in range(len(nums)):
                if ((1 << i) & sign) != 0:
                    subset.append(nums[i])
            res.append(subset)
        return res
```

CPP Code:

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int n = nums.size();
        vector<int> t;
        vector<vector<int>> res;
        for (int i = 0; i < (1 << n); ++i) {
            t.clear();
            for (int j = 0; j < n; ++j) {
                if (i & (1 << j)) {
                    t.push_back(nums[j]);
                }
            }
            res.push_back(t);
        }
        return res;
    }
};
```

JS Code:

```
var subsets = function (nums) {
  const ans = [];
  const n = nums.length;
  for (let mask = 0; mask < 1 << n; ++mask) {
    const t = [];
    for (let i = 0; i < n; ++i) {
      if (mask & (1 << i)) {
        t.push(nums[i]);
      }
    }
    ans.push(t);
  }
  return ans;
};
```

复杂度分析

令 N 为数组长度

- 时间复杂度：$O(N * 2^N)$

- 空间复杂度：$O(N)$，最长子集为整个数组长，不考虑返回结果。

上一页 下一页