18. 4Sum

Medium 2827 388 Add to List Share

Given an array `nums` of *n* integers and an integer `target`, are there elements *a*, *b*, *c*, and *d* in `nums` such that *a* + *b* + *c* + *d* = `target`? Find all unique quadruplets in the array which gives the sum of `target`.

Notice that the solution set must not contain duplicate quadruplets.

Example 1:

```
Input: nums = [1,0,-1,0,-2,2], target = 0
Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]
```

Example 2:

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

Constraints:

- `0 <= nums.length <= 200`
- `-10^9 <= nums[i] <= 10^9`
- `-10^9 <= target <= 10^9`

1. Sort
2. Exhausive attack the first number
3. Use treeSumTarget()
4. Use twoSumTarget()

Time complexity: $O(n^3)$

```cpp
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());

        vector<vector<int>> res;
        for (int i = 0; i < nums.size(); i++) {
            vector<vector<int>> triples = threeSumTarget(nums, i + 1, target - nums[i]);
            for (auto triple : triples) {
                triple.push_back(nums[i]);
                res.push_back(triple);
            }
            while (i < nums.size() - 1 && nums[i] == nums[i + 1]) {
                i++;
            }
        }
        return res;
    }

private:
    vector<vector<int>> threeSumTarget(vector<int>& nums, int start, int target) {
        vector<vector<int>> res;
        for (int i = start; i < nums.size(); i++) {
            vector<vector<int>> tuples = twoSumTarget(nums, i + 1, target - nums[i]);
            for (auto tuple : tuples) {
                tuple.push_back(nums[i]);
                res.push_back(tuple);
            }
            while (i < nums.size() - 1 && nums[i] == nums[i + 1]) {
                i++;
            }
        }
        return res;
    }

    vector<vector<int>> twoSumTarget(vector<int>& nums, int start, int target) {
        int lo = start, hi = nums.size() - 1;
        vector<vector<int>> res;
        while(lo < hi) {
            int left = nums[lo], right = nums[hi];
            int sum = left + right;
            if (target > sum) {
                while (lo < hi && nums[lo] == left) {
                    lo++;
                }
            } else if (target < sum) {
                while (lo < hi && nums[hi] == right) {
                    hi--;
                }
            } else {
                res.push_back({left, right});
                while (lo < hi && nums[lo] == left) {
                    lo++;
                }
                while (lo < hi && nums[hi] == right) {
                    hi--;
                }
            }
        }
        return res;
    }
};
```