# 题目

给你一个包含n个整数的数组nums，判断nums中是否存在三个元素a，b，c，使得a + b + c = 0？请你找出所有满足条件且不重复的三元组。

**注意：**答案中不可以包含重复的三元组。

**示例：**

给定数组nums = [-1, 0, 1, 2, -1, -4]，

满足要求的三元组集合为：

[

[-1, 0, 1],

[-1, -1, 2]

]

类似题目：

167. 两数之和 II - 输入有序数组

16. 最接近的三数之和

18. 四数之和

611. 有效三角形的个数

# 分析

## 方法一：排序+双指针

**思路：**

先排序，在排序的基础上，虽然也是O(n^2)复杂度，但可以利用双指针尽量提高效率。

**代码：**

class Solution {

public:

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

sort(nums.begin(), nums.end());

vector<vector<int>> ans;

int n = nums.size();

for (int i = 0; i < n - 2; i++) {

if (i > 0 && nums[i] == nums[i - 1]) continue; // 过滤重复的第一个数

int j = i + 1, k = n - 1;

while (j < k) {

int sum = nums[i] + nums[j] + nums[k];

if (sum < 0) {

j++;

while (j < k && nums[j] == nums[j - 1]) j++; // 过滤重复的第二个数

} else if (sum > 0) {

k--;

while (j < k && nums[k] == nums[k + 1]) k--; // 过滤重复的第三个数

} else {

ans.push\_back({nums[i], nums[j], nums[k]});

j++;

k--;

while (j < k && nums[j] == nums[j - 1]) j++; // 过滤重复的第二个数

while (j < k && nums[k] == nums[k + 1]) k--; // 过滤重复的第三个数

}

}

}

return ans;

}

};

**或：**

class Solution {

public:

vector<vector<int>> threeSum1(vector<int>& nums){

vector<vector<int>>res;

sort(nums.begin(),nums.end());

if(nums.size()<3)return res;

int l,r;

for(int i=0;i<nums.size();i++){

if(nums[i]>0)break;

//如果当前数字大于0，则三数之和一定大于0，所以结束循环

if(i>0&&nums[i]==nums[i-1])continue;//去重

l=i+1;

r=nums.size()-1;

while(l<r){

int sum=nums[i]+nums[l]+nums[r];

if(sum==0){

res.push\_back(vector<int>{nums[i],nums[l],nums[r]});

while (l<r && nums[l] == nums[l+1]) l++;

// 去重

while (l<r && nums[r] == nums[r-1]) r--;

// 去重

l++;

r--;

}

else if(sum>0)r-=1;

else l+=1;

}

}

return res;

}

};

或：

class Solution {

public:

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

sort(nums.begin(), nums.end());

vector<vector<int>> ans;

int n = nums.size();

for (int i = 0; i < n - 2; ++i) { // 遍历第一个数

int x = nums[i];

if (i && x == nums[i - 1]) continue; // 跳过重复数字

if (x + nums[i + 1] + nums[i + 2] > 0) break; // 优化一

if (x + nums[n - 2] + nums[n - 1] < 0) continue; // 优化二

int j = i + 1, k = n - 1; // 遍历弟2个数字

while (j < k) {

int s = x + nums[j] + nums[k]; // 右侧使用双指针

if (s > 0) --k;

else if (s < 0) ++j; // 双指针思想

else {

ans.push\_back({x, nums[j], nums[k]});

for (++j; j < k && nums[j] == nums[j - 1]; ++j); // 跳过重复数字

for (--k; k > j && nums[k] == nums[k + 1]; --k); // 跳过重复数字

}

}

}

return ans;

}

};

复杂度分析

时间复杂度：O(n2)，其中n为nums的长度。排序O(nlogn)。外层循环枚举第一个数，就变成167.两数之和II -输入有序数组了，做法是O(n)双指针。所以总的时间复杂度为O(n^2)。

空间复杂度：O(1)，仅用到若干变量（忽略排序的栈开销）。

## 方法二：哈希表

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

vector<vector<int>>result;

vector<int>num;

unordered\_map<int, int> M;

for (int i = 0; i < nums.size();i++){

if (M.find(nums[i]) == M.end())

{

M[nums[i]] = 1;

num.push\_back(nums[i]);

}

else

M[nums[i]]++;

}

sort(num.begin(), num.end());

int firstoutput = 1;

for (int i = 0; i < num.size(); i++)

{

for (int j = i; j < num.size();j++){

if(j==i&&M[num[j]]==1)

continue;

if(-num[i]-num[j]<num[j])

continue;

if (M.find(-num[i] - num[j]) != M.end())

{

if (-num[i] - num[j]==num[j]){

if(num[j]==num[i]){

if(M[num[j]]<=2)continue;

}

else{

if(M[num[j]]==1)continue;

}

}

result.push\_back(vector<int>{num[i],num[j],-num[i] - num[j]});

}

}

}

return result;

}