youngyangyang04 / leetcode-master

Code Issues 6 Pull requests 7 Actions Projects Wiki Security

master

leetcode-master / problems / 0015.三数之和.md

youngyangyang04 Update History

6 contributors

361 lines (300 sloc) 11.9 KB

欢迎大家参与本项目，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

> 用哈希表解决了两数之和，那么三数之和呢？

第15题. 三数之和

https://leetcode-cn.com/problems/3sum/

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。 组合

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]]

思路

注意[0， 0， 0， 0] 这组数据

哈希解法

两层for循环就可以确定 a 和b 的数值了，可以使用哈希法来确定 0-(a+b) 是否在 数组里出现过，其实这个思路是正确的，但是我们有一个非常棘手的问题，就是题目中说的不可以包含重复的三元组。

把符合条件的三元组放进vector中，然后在去去重，这样是非常费时的，很容易超时，也是这道题目通过率如此之低的根源所在。

去重的过程不好处理，有很多小细节，如果在面试中很难想到位。

时间复杂度可以做到O(n^2)，但还是比较费时的，因为不好做剪枝操作。

大家可以尝试使用哈希法写一写，就知道其困难的程度了。

哈希法C++代码:

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> result;
        sort(nums.begin(), nums.end());
        // 找出a + b + c = 0
        // a = nums[i], b = nums[j], c = -(a + b)
        for (int i = 0; i < nums.size(); i++) {
            // 排序之后如果第一个元素已经大于零，那么不可能凑成三元组
            if (nums[i] > 0) {
                continue;
            }
            if (i > 0 && nums[i] == nums[i - 1]) { //三元组元素a去重
                continue;
            }
            unordered_set<int> set;
            for (int j = i + 1; j < nums.size(); j++) {
                if (j > i + 2
                        && nums[j] == nums[j-1]
                        && nums[j-1] == nums[j-2]) { // 三元组元素b去重
                    continue;
                }
                int c = 0 - (nums[i] + nums[j]);
                if (set.find(c) != set.end()) {
                    result.push_back({nums[i], nums[j], c});
                    set.erase(c);// 三元组元素c去重
                } else {
                    set.insert(nums[j]);
                }
            }
        }
        return result;
    }
};
```

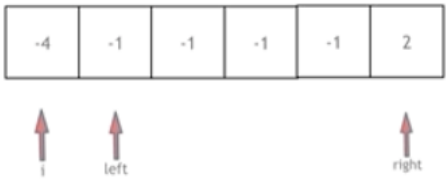
双指针

其实这道题目使用哈希法并不十分合适，因为在去重的操作中有很多细节需要注意，在面试中很难直接写出没有bug的代码。

而且使用哈希法 在使用两层for循环的时候，能做的剪枝操作很有限，虽然时间复杂度是O(n^2)，也是可以在leetcode上通过，但是程序的执行时间依然比较长 。

接下来我来介绍另一个解法：双指针法，**这道题目使用双指针法 要比哈希法高效一些**，那么来讲解一下具体实现的思路。

动画效果如下：



拿这个nums数组来举例，首先将数组排序，然后有一层for循环，i从下表0的地方开始，同时定一个下表left 定义在i+1的位置上，定义下表right 在数组结尾的位置上。

依然还是在数组中找到 abc 使得a + b +c =0，我们这里相当于 a = nums[i] b = nums[left] c = nums[right]。

接下来如何移动left 和right呢， 如果nums[i] + nums[left] + nums[right] > 0 就说明 此时三数之和大了，因为数组是排序后了，所以right下表就应该向左移动，这样才能让三数之和小一些。

如果 nums[i] + nums[left] + nums[right] < 0 说明 此时 三数之和小了，left 就向右移动，才能让三数之和大一些，直到left与right相遇为止。

时间复杂度：O(n^2)。

C++代码代码如下：

```
class Solution {    代码需要再次理解
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> result;
        sort(nums.begin(), nums.end());
        // 找出a + b + c = 0
        // a = nums[i], b = nums[left], c = nums[right]
        for (int i = 0; i < nums.size(); i++) {
            // 排序之后如果第一个元素已经大于零，那么无论如何组合都不可能凑成三元组，直接返回结果就可以了
            if (nums[i] > 0) {
                return result;
            }
            // 错误去重方法，将会漏掉-1,-1,2 这种情况
            /*
            if (nums[i] == nums[i + 1]) {
                continue;
            }
            */
            // 正确去重方法                向前去重
            if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }
            int left = i + 1;
            int right = nums.size() - 1;
            while (right > left) {
                // 去重复逻辑如果放在这里，0，0，0 的情况，可能直接导致 right<=left 了，从而漏掉了 0,0,0 这种三元组
                /*
                while (right > left && nums[right] == nums[right - 1]) right--;
                while (right > left && nums[left] == nums[left + 1]) left++;
```

```
                */
                if (nums[i] + nums[left] + nums[right] > 0) {
                    right--;
                } else if (nums[i] + nums[left] + nums[right] < 0) {
                    left++;
                } else {
                    result.push_back(vector<int>{nums[i], nums[left], nums[right]});
                    // 去重逻辑应该放在找到一个三元组之后
                    while (right > left && nums[right] == nums[right - 1]) right--;
                    while (right > left && nums[left] == nums[left + 1]) left++;

                    // 找到答案时，双指针同时收缩
                    right--;
                    left++;
                }
            }

        }
        return result;
    }
};
```

思考题

既然三数之和可以使用双指针法，我们之前讲过的1.两数之和，可不可以使用双指针法呢?

如果不能，题意如何更改就可以使用双指针法呢？ **大家留言说出自己的想法吧！**

两数之和 就不能使用双指针法，因为1.两数之和要求返回的是索引下表， 而双指针法一定要排序，一旦排序之后原数组的索引就被改变了。

如果1.两数之和要求返回的是数值的话，就可以使用双指针法了。

其他语言版本

Java：

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> result = new ArrayList<>();
        Arrays.sort(nums);

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] > 0) {
                return result;
            }

            if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }

            int left = i + 1;
```

```
            int right = nums.length - 1;
            while (right > left) {
                int sum = nums[i] + nums[left] + nums[right];
                if (sum > 0) {
                    right--;
                } else if (sum < 0) {
                    left++;
                } else {
                    result.add(Arrays.asList(nums[i], nums[left], nums[right]));

                    while (right > left && nums[right] == nums[right - 1]) right--;
                    while (right > left && nums[left] == nums[left + 1]) left++;

                    right--;
                    left++;
                }
            }
        }
        return result;
    }
}
```

Python：

```
class Solution:
    def threeSum(self, nums):
        ans = []
        n = len(nums)
        nums.sort()
        for i in range(n):
            left = i + 1
            right = n - 1
            if nums[i] > 0:
                break
            if i >= 1 and nums[i] == nums[i - 1]:
                continue
            while left < right:
                total = nums[i] + nums[left] + nums[right]
                if total > 0:
                    right -= 1
                elif total < 0:
                    left += 1
                else:
                    ans.append([nums[i], nums[left], nums[right]])
                    while left != right and nums[left] == nums[left + 1]: left += 1
                    while left != right and nums[right] == nums[right - 1]: right -= 1
                    left += 1
                    right -= 1
        return ans
```

Go：

```
func threeSum(nums []int)[][]int{
        sort.Ints(nums)
        res:=[][]int{}
```

```
        for i:=0;i<len(nums)-2;i++{
                n1:=nums[i]
                if n1>0{
                        break
                }
                if i>0&&n1==nums[i-1]{
                        continue
                }
                l,r:=i+1,len(nums)-1
                for l<r{
                        n2,n3:=nums[l],nums[r]
                        if n1+n2+n3==0{
                                res=append(res,[]int{n1,n2,n3})
                                for l<r&&nums[l]==n2{
                                        l++
                                }
                                for l<r&&nums[r]==n3{
                                        r--
                                }
                        }else if n1+n2+n3<0{
                                l++
                        }else {
                                r--
                        }
                }
        }
        return res
}
```

javaScript:

```
/**
 * @param {number[]} nums
 * @return {number[][]}
 */

// 循环内不考虑去重
var threeSum = function(nums) {
    const len = nums.length;
    if(len < 3) return [];
    nums.sort((a, b) => a - b);
    const resSet = new Set();
    for(let i = 0; i < len - 2; i++) {
        if(nums[i] > 0) break;
        let l = i + 1, r = len - 1;
        while(l < r) {
            const sum = nums[i] + nums[l] + nums[r];
            if(sum < 0) { l++; continue };
            if(sum > 0) { r--; continue };
            resSet.add(`${nums[i]},${nums[l]},${nums[r]}`);
            l++;
            r--;
        }
    }
    return Array.from(resSet).map(i => i.split(","));
};
```

```
// 去重优化
var threeSum = function(nums) {
    const len = nums.length;
    if(len < 3) return [];
    nums.sort((a, b) => a - b);
    const res = [];
    for(let i = 0; i < len - 2; i++) {
        if(nums[i] > 0) break;
        // a去重
        if(i > 0 && nums[i] === nums[i - 1]) continue;
        let l = i + 1, r = len - 1;
        while(l < r) {
            const sum = nums[i] + nums[l] + nums[r];
            if(sum < 0) { l++; continue };
            if(sum > 0) { r--; continue };
            res.push([nums[i], nums[l], nums[r]])
            // b c 去重
            while(l < r && nums[l] === nums[++l]);
            while(l < r && nums[r] === nums[--r]);
        }
    }
    return res;
};
```

ruby:

```
def is_valid(strs)
    symbol_map = {')' => '(', '}' => '{', ']' => '['}
    stack = []
    strs.size.times {|i|
        c = strs[i]
        if symbol_map.has_key?(c)
            top_e = stack.shift
            return false if symbol_map[c] != top_e
        else
            stack.unshift(c)
        end
    }
    stack.empty?
end
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录