查阅更多的题解，请点击

Problem

15. 3Sum(Medium)

Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note:
The solution set must not contain duplicate triplets.

Example:

```
Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:
[
    [-1, 0, 1],
    [-1, -1, 2]
]
```

Solution

设数组大小为n

O(n*n) time

可以按照一般套路，观察问题，显然不易reduce，则考虑其他策略。

对于算法而言，有一个很朴素的思想：**算法更容易处理结构化的数据**（这里的结构化就是无序数据->有序数据）
对于有序的输入数据，最直观的解法是：从左选定第一个数a，再在a右边的集合中选定第二个数b，再在第二个数b右边集合中查找第三个数。

然后考虑重复的triplet，三个数相同=>两个数相同。所以对于一个找到的triplet，只要其前两个数与之前的答案不同，则一定是unique的，去重策略基于这个思想，具体实现可以参考代码。

同样注意一个简单的减少查找的技巧：

- 若当前a>0 或 a+b >0，不可能存在a+b+c=0的序列（数组升序）

```
class Solution
{
public:
    vector<vector<int>> threeSum(vector<int> &nums)
    {
        vector<vector<int>> res;
        if (nums.size() < 3)
            return res;
        sort(nums.begin(), nums.end());

        if (nums.front() > 0 || nums.back() < 0)
            return res;

        unordered_map<int, int> items;
        int pos = 0, max = 0;
        for (auto item : nums)
        {
            // if(item)
            items[item] = pos++;
        }
        for (int i = 0; i < nums.size() - 2; ++i)
        {
            if (nums[i] > 0)
                break;
            for (int j = i + 1; j < nums.size() - 1; ++j)
            {
                if (nums[i] + nums[j] > 0)
                    break;
                if (items.find(0 - nums[i] - nums[j]) != items.end())
                {
                    if (items[0 - nums[i] - nums[j]] <= j)
                        break;
                    if (res.empty() || (res.back()[0] != nums[i] || res.back()[1] != nums[j]))
                        res.emplace_back(vector<int>{nums[i], nums[j], 0 - nums[i] - nums[j]});
                }
                while (j < nums.size() - 1 && nums[j] == nums[j + 1])
                    j++;
            }
            while (i < nums.size() - 2 && nums[i] == nums[i + 1])
                i++;
        }
        return res;
    }
};
```

在上述解法中，为了加速查找，用了hashtable，实际上可以有可以优化的部分：
对于选定a之后，**查找b、c的过程变成了在有序数组中，查找两个数，其和为-a**，这是一个线性时间可解的问题，用两个指针双向查找即可

```
class Solution
{
public:
    vector<vector<int>> threeSum(vector<int> &nums)
    {
        vector<vector<int>> res;
        if (nums.size() < 3)
            return res;
        sort(nums.begin(), nums.end());
        if (nums.front() > 0 || nums.back() < 0)
            return res;

        for (int i = 0; i < nums.size() - 2; ++i)
        {
            if (nums[i] > 0)
                break;
            auto target = 0 - nums[i];
            int front = i + 1, back = nums.size() - 1;
            while (front < back)
            {
                auto sum = nums[front] + nums[back];
                if (sum > target)
                    back--;
                else if (sum < target)
                    front++;
                else
                {
                    res.emplace_back(vector<int>{nums[i], nums[front++], nums[back--]});
                    while (front < back && nums[front-1] == nums[front ])
                        front++;
                    while (front < back && nums[back+1] == nums[back])
                        back--;
                }
            }
            while (i < nums.size() - 2 && nums[i] == nums[i + 1])
                i++;
        }
        return res;
    }
};
```

GitHub传送门