

我开始用的哈希表

有重复问题，案例不能通过。

```
1  class Solution {
2  public:
3      vector<vector<int>> threeSum(vector<int>& nums) {
4
5          int N = nums.size();
6          vector<vector<int>> result;
7
8          // 使用哈希表
9
10         // 定义哈希表
11         unordered_map<int, int> hashTable;
12
13         for (int i = 0; i < N; i++) {
14
15             for (int j = i + 1; j < N; j++) {
16                 int sumOfTwoNumber = nums[i] + nums[j];
17
18                 int b = -sumOfTwoNumber;
19
20                 auto it = hashTable.find(b);
21
22                 // 说明找到了
23                 if (it != hashTable.end()) {
24                     vector<int> temp;
25                     temp.push_back(it->first);
26                     temp.push_back(nums[i]);
27                     temp.push_back(nums[j]);
28                     result.push_back(temp);
29                 }
30             }
31
32             hashTable[nums[i]] = i;
33         }
34
35         return result;
36     }
37 };`
```

去重复后的哈希表

```
```c++
```

```
class Solution {
public:
 vector<vector<int>> threeSum(vector<int>& nums) {

 int N = nums.size();
 vector<vector<int>> result;

 // 使用哈希表
 unordered_set<string> seen;

 // 用于存储已找到的三元组，避免重复
 unordered_map<int, int> hashTable;

 for (int i = 0; i < N; i++) {
 for (int j = i + 1; j < N; j++) {
 int sumOfTwoNumber = nums[i] + nums[j];
 int b = -sumOfTwoNumber;

 auto it = hashTable.find(b);
 // 说明找到了
 if (it != hashTable.end() && it->second < i) {
 // 创建三元组并排序以避免重复
 vector<int> temp = {it->first, nums[i],
nums[j]};

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

 // 使用字符串作为唯一标识
 string key = to_string(temp[0]) + "," +
to_string(temp[1]) +
 "," + to_string(temp[2]);
 if (seen.find(key) == seen.end()) {
 result.push_back(temp);
 seen.insert(key);
 }
 }
 }
 }

 // 记录当前数字的位置
 hashTable[nums[i]] = i;
 }
}
```

```

84 return result;
85 }
86 };

```

## 标准题解（双指针）

```

1 class Solution {
2 public:
3 vector<vector<int>> threeSum(vector<int>& nums) {
4 sort(nums.begin(), nums.end());
5 // 待返回的三元组
6 vector<vector<int>> triples;
7
8 for (int i = 0; i < nums.size(); ++i) {
9 // 检测重复的 nums[i]
10 if (i > 0 && nums[i] == nums[i - 1]) continue;
11
12 int l = i + 1;
13 int r = nums.size() - 1;
14
15 // 不同处，因为每个指令执行之前均会考虑 while (l < r)，所以
 // 不用考虑越界或者错位
16 while (l < r) {
17 // 检测重复的 nums[l]
18 if (l > i + 1 && nums[l] == nums[l - 1]) {
19 ++l;
20 }
21 // 检测重复的 nums[r]
22 else if (r < nums.size() - 1 && nums[r] == nums[r
+ 1]) {
23 --r;
24 }
25 // 均不重复再按照两数之和的思路
26 else if (nums[i] + nums[l] + nums[r] > 0) {
27 --r;
28 }
29 else if (nums[i] + nums[l] + nums[r] < 0) {
30 ++l;
31 } else {
32 triples.push_back({nums[i], nums[l],
nums[r]});
33 ++l; --r;
34 }
35 }
36 }

```

```
37
38 return triples;
39 }
40 };
```