散列表：（Hash table，也叫哈希表），是根据键（Key）而直接访问在内存存储位置的数据结构。也就是说，它通过计算一个关于键值的函数，将所需查询的数据映射到表中一个位置来访问记录，这加快了查找速度。这个映射函数称做散列函数，存放记录的数组称做散列表。

散列函数: (HashFunction)：若关键字为k，则其值存放在f(k)的存储位置上。由此，不需比较便可直接取得所查记录。称这个对应关系为散列函数

哈希碰撞：对不同的关键字可能得到同一散列地址k1≠k2，而f(k1)=f(k2)，这种现象称为冲突

Map（常用于查找数值和计数）

set

HashMap HashSet ：两者的底层是哈希表的实现，查找的时间复杂度为O（1）

TreeSet treeMap ：两者的底层是树的实现，里面的元素是经过排序的，查找的时间复杂度为o(logn)

例题：

1、给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。（https://leetcode-cn.com/problems/valid-anagram/ ）

//method1:Sort the two Strings and judge whether they are equal

public boolean isAnnagram(String s,String t) {

```
         if(s==null||t==null) return false;
         //If the length of two Strings are different,the result is false.
        if(s.length()!=t.length()) return false;
        char[] s1=s.toCharArray();
        char[] t1=t.toCharArray();

        Arrays.sort(s1);
        Arrays.sort(t1);
        return  Arrays.equals(s1,t1);

}

//method2: count the number of characters of two Strings and

```

// 时间复杂度：O(n)O(n)。时间复杂度为 O(n)O(n) 因为访问计数器表是一个固定的时间操作。

// 空间复杂度：O(1)O(1)。尽管我们使用了额外的空间，但是空间的复杂性是 O(1)O(1)，因为无论 NN 有多大，表的大小都保持不变。

```
public boolean isAnnagram2(String s,String t) {

         if(s==null||t==null) return false;
         //If the length of two Strings are different,the result is false.
```

```
        if(s.length()!=t.length()) return false;

        int[] counter=new int[26];
       for(int i=0;i<s.length();i++){
           counter[s.charAt(i)-'a']++;
           counter[t.charAt(i)-'a']--;
       }
       for(int count:counter){
           if(count!=0) return false ;
       }
       return true;

}
```

2、两数之和：给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那 两个 整数，并返回他们的数组下标。你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。（ https://leetcode-cn.com/problems/two-sum/ ）

```
public class TwoSum {
public int[] twoSum(int[] nums,int target){
if(nums==null || nums.length<2) return null;
Map<Integer,Integer> map=new HashMap();
for(int i=0;i<nums.length;i++){
map.put(nums[i],i);
}
for (int i=0;i<nums.length;i++) {
int temp=target-nums[i];
if(map.containsKey(temp)&&map.get(temp)!=i){
return new int[]{i,map.get(temp)};
}
}
return null;
}
}
```

3、三数之和：给定一个包含 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 ？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。（ https://leetcode-cn.com/problems/3sum/ ）

```
class Solution {
```

```
public static List<List<Integer>> threeSum(int[] nums) {
List<List<Integer>> ans = new ArrayList();
int len = nums.length;
if(nums == null || len < 3) return ans;
Arrays.sort(nums); // 排序
for (int i = 0; i < len ; 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
int L = i+1;
int R = len-1;
while(L < R){
int sum = nums[i] + nums[L] + nums[R];
if(sum == 0){
ans.add(Arrays.asList(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) L++;
else if (sum > 0) R--;
}
}
return ans;
}
```
