## 哈希表(HashMap):是根据关键码的值而进行直接访问的数据结构

**比如**：数组就是一张哈希表

**作用**：用来判断**一个元素**是否出现在**哈希表集合**里面

**映射关系**：哈希函数

**哈希碰撞的解决方案**：拉链法和线性探测法（tableSize大于dataSize）

## 三种常见的哈希结构

对于集合:

1、优先使用unordered_set（查询和增删的速度都快）

2、然后使用set(如果要求集合有序)

3、使用multiset(要求集合有序且有重复数据)

map的简单使用：

## 通过例题说明几个道理:

### (1)数组就是哈希表

第242题. 有效的字母异位词

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。

示例 1:

```
输入: s = "anagram", t = "nagaram"
输出: true
```

示例 2:

```
输入: s = "rat", t = "car"
输出: false
```

代码随想录

「说明:」你可以假设字符串只包含小写字母。

> 只要一个数组record,其数组大小为26,其key值就是索引，因为字符a到字符z的ASCII也是26个连续的数值。
> 作用记录字符串s和t出现某字符的次数(在s字符串中出现一次加一，在t字符串中出现一次减一)

### （2）如果哈希值过大，还是得用Set

> 如果**哈希值比较大**，**数量特别少**，**跨度非常大**，使用数组就造成空间的极大浪费，此时用set

## 第349题. 两个数组的交集

题意：给定两个数组，编写一个函数来计算它们的交集。

**示例 1：**

```
输入：nums1 = [1,2,2,1], nums2 = [2,2]
输出：[2]
```

**示例 2：**

```
输入：nums1 = [4,9,5], nums2 = [9,4,9,8,4]
输出：[9,4]
```



**「说明：」**

输出结果中的每个元素一定是唯一的。

我们可以不考虑输出结果的顺序。

> 1:由于数值的范围非常大不适合用数组
> 2:由于输出的结果不可重复
> 所以用set或者但是unordered_set又因为unordered_set的效率比set大

```
# include<unordered_set>
# include<vector>
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
 unordered_set<int> numsSet(nums1.begin(),nums.end());
 unordered_set<int> ansSet;
 for(int i=0;i<nums2.size();i++){
 if(numsSet.find(nums2[i])!=numsSet.end()){
  ansSet.insert(nums2[i]);
  }
  }
  return vector<int>(ansSet.begin(),ansSet.end());
     }
};
```

# 第202题. 快乐数

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」定义为：对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和，然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。如果 可以变为 1，那么这个数就是快乐数。

如果 n 是快乐数就返回 True ；不是，则返回 False 。

> 由于题目中点出会出现无线循环，如果出现无线循环即每个位置上的数字平方和会重复即可以用一个哈希表存储每次计算出现的sum，如果出现重复的sum则循环结束失败.
> **「当我们遇到了要快速判断一个元素是否出现集合里的时候，就要考虑哈希法了。」**

```
class Solution {
public:
    bool isHappy(int n) {
        unordered_set<int> sumSet;
        int sum=0;
        while(1){
            if(n==1) return true;
            while(n){
                sum+=pow(n%10,2);
                 n/=10;
             }
             if(sumSet.find(sum)!=sumSet.end()) return false;
             sumSet.insert(sum);
             n=sum;
             sum=0;
         }
     }
};
```

## (3)哈希法中只用数组和set还是不够所以需要用到map

# 第1题. 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，数组中同一个元素不能使用两遍。

「示例:」

给定 nums = [2, 7, 11, 15], target = 9
因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]

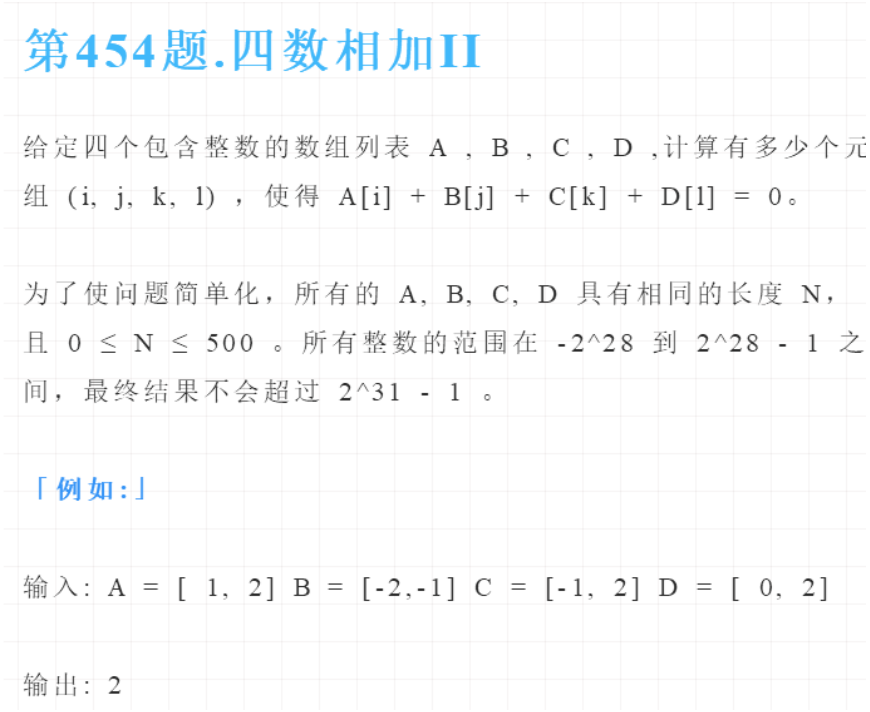
> 由于本题中需要返回他们的数组下标所以不能用set和数组只有一个值
>
> - 数组的大小是受限制的，而且如果元素很少，而哈希值太大会造成内存空间的浪费。
> - set是一个集合，里面放的元素只能是一个key，而两数之和这道题目，不仅要判断y是否存在而且还要记录y的下表位置，因为要返回x 和 y的下表。所以set 也不能用。

**注：优先考虑unordered_map,如果要求有序则用map,如果要求重复元素且有序用multimap**

```
//本道题考虑的加入unordered_map的顺序非常有趣
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
 unordered_map<int,int> map;//第一个int为数组下标,第二个int为数组值
        for(int i=0;i<nums.size();i++){
            if(map.find(target-nums[i])!=map.end()){
                return {i,map.find(target-nums[i])->second};
            }
             map.insert(make_pair(nums[i],i));
         }
         return {};
    }
```

```
};
```

## (4)其实需要哈希的地方都能找到map的身影



第454题.四数相加II

给定四个包含整数的数组列表 A , B , C , D ,计算有多少个元组 (i, j, k, l) ，使得 A[i] + B[j] + C[k] + D[l] = 0。

为了使问题简单化，所有的 A, B, C, D 具有相同的长度 N，且 0 ≤ N ≤ 500 。所有整数的范围在 -2^28 到 2^28 - 1 之间，最终结果不会超过 2^31 - 1 。

「例如:」

输入: A = [ 1, 2] B = [-2,-1] C = [-1, 2] D = [ 0, 2]

输出: 2

> 思路:
> A[i]+B[j]为一个map<int,int>第一个int表示相加结果，第二个int为相对应的相加结果出现的次数
> 同理C[k]+D[l]为一个map<int,int>

```
class Solution {
public:
    int fourSumCount(vector<int>& nums1, vector<int>& nums2, vector<int>& ums3, vector<int>& nums4) {
  long long ans=0;
  unordered_map<int,int>leftMap;
  unordered_map<int,int>rightMap;
  for(int i=0;i<nums1.size();i++){
  for(int j=0;j<nums2.size();j++){
  if(leftMap.find(nums1[i]+nums2[j])!=leftMap.end()){
   leftMap.find(nums1[i]+nums2[j])->second+=1;
   }else{
   leftMap.insert(make_pair(nums1[i]+nums2[j],1));
   }
   }
   }
   for(int i=0;i<nums3.size();i++){
   for(int j=0;j<nums4.size();j++){
   if(rightMap.find(nums3[i]+nums4[j])!=rightMap.end()){
```

```
19   rightMap.find(nums3[i]+nums4[j])->second+=1;
20   }else{
21   rightMap.insert(make_pair(nums3[i]+nums4[j],1));
22   }
23   }
24   }
25   unordered_map<int,int>::iterator it=leftMap.begin();
26   while(it!=leftMap.end()){
27   if(rightMap.find(-1*(it->first))!=rightMap.end()){
28   ans+=(it->second)*(rightMap.find(it->first)->second);
29   }
30   }
31   return ans;
32      }
33  };
```

以上的做法会超时限制

```
1  class Solution {
2  public:
3      int fourSumCount(vector<int>& nums1, vector<int>& nums2, vector<int>&
ums3, vector<int>& nums4) {
4              int ans=0;
5              unordered_map<int,int>leftMap;//记录A+B,前者为相加结果,后者为出
现次数
6              for(int i=0;i<nums1.size();i++){
7                  for(int j=0;j<nums2.size();j++){
8                      if(leftMap.find(nums1[i]+nums2[j])!=leftMap.end()){
9                          leftMap.find(nums1[i]+nums2[j])->second+=1;
10                      }else{
11                          leftMap.insert(make_pair(nums1[i]+nums2[j],1));
12                      }
13                  }
14              }
15              for(int i=0;i<nums3.size();i++){
16                  for(int j=0;j<nums4.size();j++){
17                      if(leftMap.find(-1*
(nums3[i]+nums4[j]))!=leftMap.end()){
18                          ans+=(leftMap.find(-1*(nums3[i]+nums4[j]))->seco
nd);
19                      }
20                  }
21              }
```

```
22              return ans;
23      }
24  };
```

**(4)在哈希法中**

**有一些场景就是为数组量身定做的**

## 第383题. 赎金信

给定一个赎金信 (ransom) 字符串和一个杂志(magazine)字符串，判断第一个字符串 ransom 能不能由第二个字符串 magazines 里面的字符构成。如果可以构成，返回 true ；否则返回 false。

(题目说明：为了不暴露赎金信字迹，要从杂志上搜索各个需要的字母，组成单词来表达意思。杂志字符串中的每个字符只能在赎金信字符串中使用一次。)

「注意：」

你可以假设两个字符串均只含有小写字母。

canConstruct("a", "b") -> false canConstruct("aa", "ab") -> false
canConstruct("aa", "aab") -> true

# 总结：

对于哈希三个结构:

**数组:**

key就是索引，value就是元素值，如果key的跨度不大且比较密集可以采用数组，否则采用map（存在且多少的问题）

**集合:**

首先考虑**unordered_set**，如果追求有序则用**set**，如果有序且可重复则用**multiset**

key就是元素值，(存在与否问题)

**映射:**

首先考虑**unordered_map**,如果追求有序则用**map**,如果有序且可重复则用**multimap**

(存在且多少的问题)