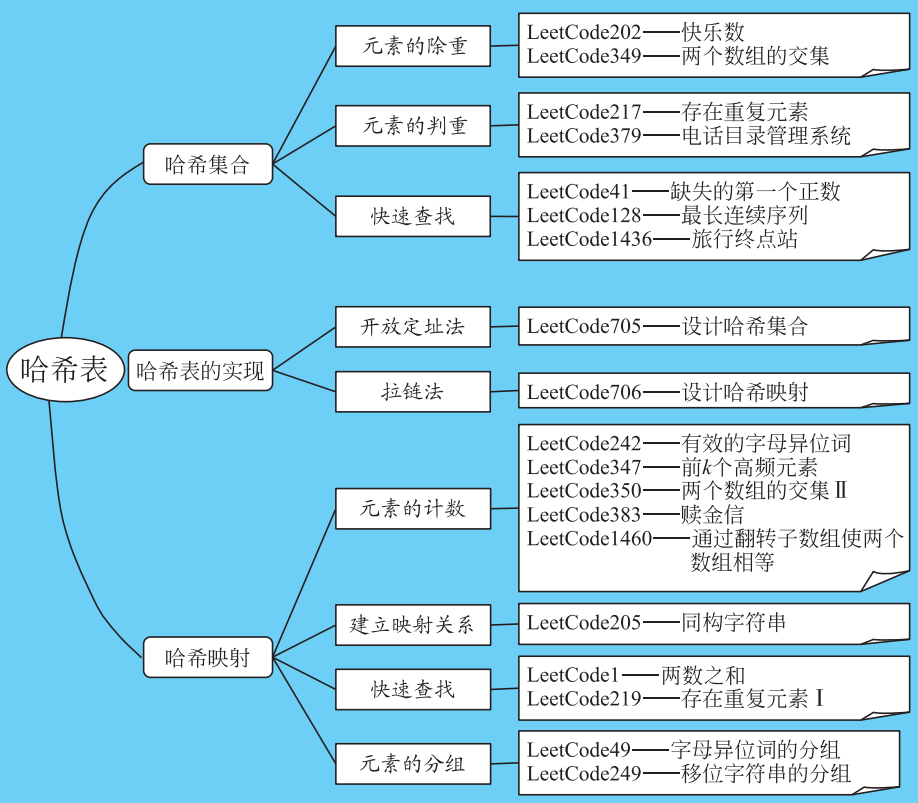
第5章 哈希表

知识图谱


哈希表
哈希集合
元素的除重
LeetCode202——快乐数
LeetCode349——两个数组的交集
元素的判重
LeetCode217——存在重复元素
LeetCode379——电话目录管理系统
快速查找
LeetCode41——缺失的第一个正数
LeetCode128——最长连续序列
LeetCode1436——旅行终点站
哈希表的实现
开放定址法
LeetCode705——设计哈希集合
拉链法
LeetCode706——设计哈希映射
哈希映射
元素的计数
LeetCode242——有效的字母异位词
LeetCode347——前k个高频元素
LeetCode350——两个数组的交集Ⅱ
LeetCode383——赎金信
LeetCode1460——通过翻转子数组使两个数组相等
建立映射关系
LeetCode205——同构字符串
快速查找
LeetCode1——两数之和
LeetCode219——存在重复元素Ⅰ
元素的分组
LeetCode49——字母异位词的分组
LeetCode249——移位字符串的分组

5.1 哈希表概述

5.1.1 哈希表的定义

1. 哈希表和哈希函数

哈希表也称为散列表，是一种根据键直接进行访问的数据结构，也就是通过把键映射到表中的一个位置来访问元素，以加快查找的速度。这个映射函数称为哈希函数，存放元素的数组称为哈希表。

假设哈希表为 ht[0..MAXH−1]（长度为 MAXH），哈希函数为 h。给定一个元素集合，每个元素的键是唯一的，以每个元素的键 k 为自变量，通过一个哈希函数 h 把 k 映射为内存单元的地址（或相对地址）$h(k)$，并把该元素存储在这个内存单元中。

这样，对于两个不同的键 k_i 和 k_j（$i \neq j$）可能出现 $h(k_i)=h(k_j)$，这种现象称为哈希冲突，将具有不同键但哈希函数值相同的元素称为“同义词”，这种冲突也称为同义词冲突。

若对于元素集合中的任意一个元素，经哈希函数映射到哈希地址集合中任何一个地址的概率是相等的，则称该哈希函数为均匀哈希函数，这就是使键经过哈希函数得到一个“随机的地址”，从而减少哈希冲突。

设计哈希函数的常用方法之一是除留余数法，该方法取键被某个不大于哈希表长度 MAXH 的整数 p 除后所得的余数为哈希地址，即 $h(k)=k \% p$（$p \leqslant$MAXH），对 p 的选择很重要，若 p 选的不好容易产生同义词，一般取小于或等于 MAXH 的素数。如图 5.1 所示为一个哈希表示意图。

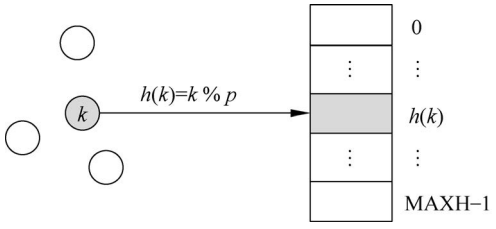


图 5.1　哈希表示意图

2. 解决哈希冲突的方法

解决哈希冲突的方法有许多，可分为开放定址法和拉链法两大类。

1）开放定址法

开放定址法就是在插入一个键为 k 的元素时，若发生哈希冲突，通过某种哈希冲突解决函数（也称为再哈希）得到一个新空闲地址再插入该元素的方法。这样的哈希冲突解决函数设计有很多种，下面介绍常用的两种。

（1）线性探测法。线性探测法是从发生冲突的地址（设为 d_0）开始，依次探测 d_0 的下一个地址，直到找到一个空闲单元为止。线性探测法的迭代公式为：

$$d_0 = h(k)$$

$$d_i = (d_{i-1} + 1)\%\text{MAXH} \quad (1 \leqslant i \leqslant \text{MAXH} - 1)$$

其中，模 MAXH 是为了保证试探 0～MAXH－1 的有效地址，当到达地址为 MAXH－1 的哈希表的表尾时，下一个探测的地址是表首地址 0。

(2) 平方探测法。设发生冲突的地址为 d_0，平方探测法的探测序列为 d_0+1^2，d_0-1^2，d_0+2^2，d_0-2^2，…。平方探测法的数学描述公式为：

$$d_0 = h(k)$$

$$d_i = (d_0 \pm i^2)\ \%\ \text{MAXH} \quad (1 \leqslant i \leqslant \text{MAXH} - 1)$$

线性探测法是沿着一个方向循环试探下一个地址，平方探测法是前后试探下一个地址。此外，开放定址法的探测方法还有伪随机序列法、双哈希函数法等。

例如，某个哈希表 ht 的长度 MAXH＝8，哈希函数为 $h(k)=k\%7$，空位置用键 NULLKEY(－1)表示，删除位置用键 DELKEY(－2)表示，使用线性探测法解决冲突，求依次插入 2、9、16、15 和 1，删除 9，再插入 8 和 16 的结果，并求最终哈希表的成功查找和不成功查找的平均查找长度。创建哈希表的过程如下。

(1) 初始哈希表如图 5.2(a)所示。

(2) 插入 2，$h(2)=2\%7=2$，$d_0=2$，插入 ht[2]中(探测一次)；插入 9，$h(9)=9\%7=2$，$d_0=2$，冲突(ht[2]已经被 2 占用)，使用线性探测法，求出 $d_1=(d_0+1)\%8=3$，插入 ht[3]中(探测两次)；插入 16，$h(16)=16\%7=2$，$d_0=2$，冲突，使用线性探测法，求出 $d_1=(d_0+1)\%8=3$，仍冲突，求出 $d_2=(d_1+1)\%8=4$，插入 ht[4]中(探测 3 次)，如图 5.2(b)所示。

(3) 插入 15，$h(15)=15\%7=1$，$d_0=1$，插入 ht[1]中(探测一次)；插入 1，$h(1)=1\%7=1$，$d_0=1$，冲突，使用线性探测法，求出 $d_1=(d_0+1)\%8=2$，仍冲突，一直到 $d_4=5$，插入 ht[5]中(探测 5 次)，如图 5.2(c)所示。

(4) 删除 9，找到 ht[3]＝9，将该位置的键改为 DELKEY，探测次数置为 0，如图 5.2(d)所示。

(5) 插入 8，$h(8)=8\%7=1$，$d_0=1$，冲突，使用线性探测法，求出 $d_1=(d_0+1)\%8=2$，仍冲突，$d_2=(d_1+1)\%8=3$，ht[3]的键为 DELKEY，将 8 插入 ht[3]中(探测 3 次)，插入 16，在 ht 中找到 ht[4]＝16，说明键重复，不能插入，如图 5.2(e)所示。

此时哈希表中包含 $n=5$ 个键，其查找成功的平均查找长度($\text{ASL}_{\text{成功}}$)为 5 个键的平均探测次数，即：

$$\text{ASL}_{\text{成功}} = \frac{1+1+3+3+5}{5} = 2.6$$

现在求查找不成功的平均查找长度($\text{ASL}_{\text{不成功}}$)，假设给定键 x，它不在哈希表中，也需要按照查找过程找到空位置为止。分析哈希函数可知 $h(x)$可能的取值为 0～6。

(1) 若 $h(x)=0$，探测一次确定查找失败。

(2) 若 $h(x)=1$，探测 6 次(依次与 ht[1..6]比较)确定查找失败。

(3) 若 $h(x)=2$，探测 5 次确定查找失败。

(4) 若 $h(x)=3$，探测 4 次确定查找失败。

(5) 若 $h(x)=4$，探测 3 次确定查找失败。

(6) 若 $h(x)=5$，探测两次确定查找失败。

(7) 若 $h(x)=5$，探测一次确定查找失败。

查找不成功的平均查找长度就是所有可能查找失败的平均探测次数，即：

$$ASL_{不成功}=\frac{1+6+5+4+3+2+1}{7}=3.14$$

地址	0	1	2	3	4	5	6	7
关键字	-1	-1	-1	-1	-1	-1	-1	-1
探测次数	0	0	0	0	0	0	0	0

(a) 初始哈希表

地址	0	1	2	3	4	5	6	7
关键字	-1	-1	2	9	16	-1	-1	-1
探测次数	0	0	1	2	3	0	0	0

(b) 插入2、9、16的结果

地址	0	1	2	3	4	5	6	7
关键字	-1	15	2	9	16	1	-1	-1
探测次数	0	1	1	2	3	5	0	0

(c) 插入15和1的结果

地址	0	1	2	3	4	5	6	7
关键字	-1	15	2	-2	16	1	-1	-1
探测次数	0	1	1	0	3	5	0	0

(d) 删除9的结果

地址	0	1	2	3	4	5	6	7
关键字	-1	15	2	8	16	1	-1	-1
探测次数	0	1	1	3	3	5	0	0

(e) 插入8和16的结果

图 5.2　哈希表的操作过程(1)

2) 拉链法

拉链法是把所有的同义词用单链表链接起来的方法(每个这样的单链表称为一个桶)。如图 5.2 所示，哈希函数为 $h(k)=k$ % MAXH，所有哈希地址为 i 的元素对应的结点构成一个单链表，称为单链表 i。哈希表的地址空间为 0～MAXH－1，在地址 i 的单元中存放对应单链表的首结点地址。

例如，某个哈希表 ht 的长度 MAXH＝5，哈希函数为 $h(k)=k\%5$，使用拉链法解决冲突，求依次插入 2、9、16、15、1、6，删除 1 和 9，再插入 8 和 7 的结果，并求最终哈希表的成功查找和不成功查找的平均查找长度。创建哈希表的过程如下。

(1) 初始哈希表如图 5.3(a)所示。

(2) 插入 2、9、16、15、1、6，求出各键的哈希函数值，$h(2)=2$，$h(9)=4$，$h(16)=1$，$h(15)=0$，$h(1)=1$，$h(6)=1$。对于 $h(k)=i$ 的元素创建一个存放结点 p，使用头插法插入单链表 i 中，按上述顺序插入得到的哈希表如图 5.3(b)所示。

(3) 删除 1，$h(1)=1$，在单链表 1 中找到键为 1 的结点，删除之；删除 9，$h(9)=4$，在单链表 4 中找到键为 9 的结点，删除之，删除后的哈希表如图 5.3(c)所示。

(4) 插入 8，$h(8)=3$，将其插入单链表 3 中；插入 7，$h(7)=2$，将其插入单链表 2 的头部，如图 5.3(d)所示。

此时哈希表中包含 $n=6$ 个结点，查找 15、6、7、8 均需要比较一次，查找 16 和 2 均需要比较两次，其查找成功的平均查找长度为：

$$ASL_{成功}=\frac{1\times4+2\times2}{6}=1.3$$

现在求查找不成功的平均查找长度($ASL_{不成功}$)，假设给定键 x，它不在哈希表中，也需要按

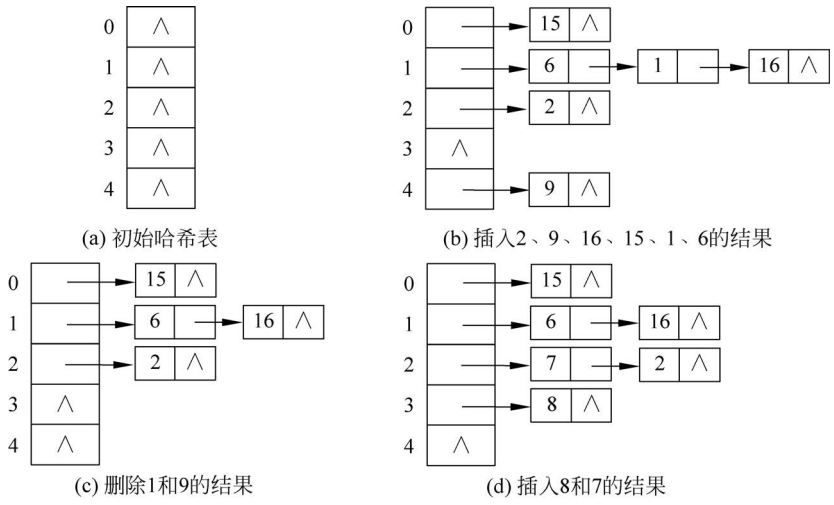


图 5.3 哈希表的操作过程(2)

照查找过程找到空位置为止。分析哈希函数可知 $h(x)$ 可能的取值为 0～4。

(1) 若 $h(x)=0$,单链表 0 的长度为 1,探测一次确定查找失败。

(2) 若 $h(x)=1$,单链表 1 的长度为 2,探测两次确定查找失败。

(3) 若 $h(x)=2$,单链表 2 的长度为 2,探测两次确定查找失败。

(4) 若 $h(x)=3$,单链表 3 的长度为 1,探测一次确定查找失败。

(5) 若 $h(x)=4$,单链表 4 的长度为 0,探测 0 次确定查找失败。

对应的查找不成功的平均查找长度为:

$$ASL_{不成功}=\frac{1+2+2+1+0}{5}=1.2$$

一般来说,使用拉链法解决冲突的哈希表的查找性能更好,但占用的空间较多。目前,C++和 Python 等语言中提供的哈希表均用拉链法或者改进方法实现,查找时间可以接近 $O(1)$。

5.1.2 哈希表的知识点

在实现哈希表时其中元素的类型通常分为两种,一种是每个元素含单个键 key,这样的哈希表称为哈希集合;另一种是每个元素为键值对(key,value),这样的哈希表称为哈希映射。通常,哈希集合用于除重、判重和快速查找,哈希映射用于计数、判重、分组数据存储和快速查找等。

1. C++中的哈希集合

C++ STL 提供的哈希集合为 unordered_set 类模板,每个元素为一个键,所有键是唯一的。unordered_set 用哈希表实现,使用拉链法解决冲突,所有元素是无序排列的,按键查找的时间复杂度接近 $O(1)$。unordered_set 的主要成员函数如下。

- empty():判断哈希集合是否为空。
- size():返回哈希集合的长度。
- [k]:返回哈希集合中键为 k 的元素。

- find(k)：如果哈希集合中存在键为 k 的元素，返回其迭代器，否则返回 end()。
- count(k)：返回哈希集合中键 k 出现的次数，结果为 0 或者 1。
- insert(e)：在哈希集合中插入元素 e。
- emplace(e)：在哈希集合中插入元素 e，比 insert(e)的效率更高。
- erase()：从哈希集合中删除一个或者几个元素。
- clear()：删除哈希集合中的所有元素。
- begin()：返回哈希集合中首元素的迭代器。
- end()：返回哈希集合中尾元素的后一个位置的迭代器。

C++：

```
unordered_set<int> hset;                    //定义元素类型为 int 的哈希集合
hset.insert(3);                             //插入 3
hset.insert(1);                             //插入 2
hset.insert(2);                             //插入 2
printf("%d\n",hset.count(1));               //输出:1
hset.erase(1);                              //删除 1
for(auto x:hset)                            //输出:2 3
    printf("%d ",x);
printf("\n");
```

2. C++中的哈希映射

C++ STL 提供的哈希映射为 unordered_map 类模板，每个元素为 pair 类型，pair 结构类型的声明如下：

```
struct pair {
    T1 first;                               //关键字成员
    T2 second;                              //值成员
};
```

其中，first 对应 key，要求所有元素的键是唯一的，second 对应值 value。同时，pair 对==、!=、<、>、<=、>=共 6 个运算符进行重载，提供了按照字典序对元素对进行大小比较的比较运算符模板函数。

unordered_map 用哈希表实现，使用拉链法解决冲突，所有元素是无序排列的，按键查找的时间复杂度接近 $O(1)$。unordered_map 的主要成员函数如下。

- empty()：判断哈希映射是否为空。
- size()：返回哈希映射中实际的元素个数。
- map[k]：返回键为 k 的元素，当其不存在时以 k 作为键插入一个元素。
- at[k]：同 map[k]。
- find(k)：在哈希映射中查找键为 k 的元素。
- count(k)：返回哈希映射中键为 k 的元素的个数，结果为 1 或者 0。
- insert(e)：在哈希映射中插入一个元素 e 并返回该元素的位置。
- emplace(e)：在哈希映射中插入元素 e，比 insert(e)的效率更高。
- erase()：删除哈希映射中的一个或者几个元素。
- clear()：删除哈希映射中的所有元素。
- begin()：返回哈希映射中首元素的迭代器。

- end()：返回哈希映射中尾元素的后一个位置的迭代器。

C++：

```
unordered_map<string,int> hmap;                    //定义键为 string、值为 int 的哈希映射
hmap.insert(pair<string,int>("Mary",3));           //插入("Mary",3)
hmap["Smith"]=1;                                   //插入("Smith",1)
hmap.insert(make_pair<string,int>("John",2));      //插入("John",2)
printf("%d\n",hmap.count("Smith"));                //输出:1
hmap.erase("Mary");                                //删除 Mary 的元素
for(auto x:hmap)                                   //输出:[John,2] [Smith,1]
    cout << "[" << x.first << "," << x.second << "] ";
cout << endl;
```

3. Python 中的哈希集合

在 Python 中提供了集合类型，用哈希表实现，在集合中存放不可变类型数据，如字符串、数字或者元组。集合是一个无序的不重复元素序列，其基本功能包括关系测试和消除重复元素。两个集合 a 和 b 之间可以做－、|、& 和^运算，其中 a－b 返回 a 中包含但 b 中不包含的元素的集合，a | b 返回 a 或 b 中包含的所有元素的集合，a & b 返回 a 和 b 中都包含的元素的集合，a^b 返回不同时包含于 a 和 b 的元素的集合。

使用大括号{ }或者 set()函数创建集合，但创建一个空集合必须用 set()而不是用{ }，因为{ }用来创建一个空字典。集合 s 的基本操作如下。

- len(s)：返回集合 s 中元素的个数。
- x in s：若元素 x 在集合 s 中，返回 True，否则返回 False。
- x not in s：若元素 x 不在集合 s 中，返回 True，否则返回 False。
- s.add(x)：向集合 s 中添加元素 x。
- s.clear()：移除集合 s 中的所有元素。
- s.isdisjoint(t)：判断两个集合 s 和 t 是否不存在交集，若不存在返回 True，否则返回 False。
- s.issubs(t)：判断集合 s 是否为集合 t 的子集。
- s.remove(x)：移除集合 s 中的元素 x，如果元素 x 不存在，则会发生错误。
- s.discard(x)：移除集合 s 中的元素 x，如果元素 x 不存在，不会发生错误。
- s.update(x)：将 x 添加到集合 s 中，且参数可以是列表、元组或者字典等。
- s.difference(t)：返回两个集合 s 和 t 的差集，即 s－t。
- s.intersection(t)：返回两个集合 s 和 t 的交集，即 s & t。
- s.union(t)：返回两个集合 s 和 t 的并集，即 s | t。
- s.symmetric_difference(t)：返回除集合 s 和集合 t 共有的元素以外的元素，即 s^t。

Python：

```
hset=set()              #定义元素类型为 int 的哈希集合
hset.add(3)             #插入 3
hset.add(1)             #插入 2
hset.add(2)             #插入 2
print(1 in hset)        #输出:True
hset.remove(1)          #删除 1
for x in hset:          #输出:2 3
    print(x,end=' ')
print()
```

4. Python 中的哈希映射

Python 中提供的字典类型相当于哈希映射，也是用哈希表实现。字典可以存储任意类型的对象，每个元素由 key：value 构成，其中 key 是键，value 是对应的值，中间用逗号分隔，整个字典包含在大括号({ })中，键必须是唯一的，但值不必唯一。值可以取任何数据类型，但键必须是不可变的数据类型，如字符串、数字或者元组。

使用大括号{ }或者 dict()函数创建字典。字典 dict 的基本操作如下。

- len(dict)：返回字典 dict 中元素的个数。
- dict[key]：返回字典 dict 中键 key 的值。
- dict.clear()：移除字典 dict 中的所有元素。
- key in dict：如果键 key 在字典 dict 中，返回 True，否则返回 False。
- keynot in dict：如果键 key 不在字典 dict 中，返回 True，否则返回 False。
- dict.has_key(key)：如果键 key 在字典 dict 中，返回 True，否则返回 False。
- dict.items()：以列表形式返回字典 dict 中可遍历的(键，值)元组数组。
- dict.keys()：以列表形式返回字典 dict 中的所有键。
- dict.values()：以列表形式返回字典 dict 中的所有值。
- dict.get(key, default=None)：返回字典 dict 中指定键的值，如果值不在字典中，则返回 default 值。
- dict.setdefault(key,default=None)：和 get()类似，但如果键不在字典中，将会添加键，并将值设置为 default。
- pop(key[,default])：删除字典 dict 中键 key 对应的值，返回值为被删除的值。key 值必须给出，否则将返回 default 值。
- popitem()：随机返回并删除字典 dict 中的最后一对键值。

Python：

```
hmap=dict()                          #定义一个哈希字典
hmap["Mary"]=3                       #插入("Mary",3)
hmap["Smith"]=1                      #插入("Smith",1)
hmap["John"]=2                       #插入("John",2)
print("Smith" in hmap)               #输出:True
del hmap["Mary"]                     #删除 Mary 的元素
for name,v in hmap.items():          #输出:[Smith,1] [John,2]
    print("[%s,%d]"%(name,v),end=' ')
print()
```

另外，Python 提供了字典 dict 的一个子类 defaultdict，用 defaultdict 函数创建该子类的对象。以下语句定义一个 defaultdict 类对象 dict1：

```
dict1=defaultdict(factory_function)
```

其中，factory_function 可以是 list、set、str 等，作用是当 key 不存在时返回工厂函数的默认值，如 list 对应[ ]、str 对应空字符串、set 对应 set()、int 对应 0，也就是说 dict1 会为一个不存在的键提供默认值，从而避免 KeyError 异常。dict1 的其他功能与 dict 相同。

5.2 哈希表的实现

5.2.1 LeetCode705——设计哈希集合★

【问题描述】 不使用任何内建的哈希表库设计一个哈希集合，实现 MyHashSet 类。

(1) void add(key)：向哈希集合中插入值 key。

(2) bool contains(key)：返回哈希集合中是否存在这个值 key。

(3) void remove(key)：将给定值 key 从哈希集合中删除，如果其中没有这个值，什么也不做。

示例：

```
MyHashSet myHashSet = new MyHashSet();
myHashSet.add(1);                    //set=[1]
myHashSet.add(2);                    //set=[1, 2]
myHashSet.contains(1);               //返回 true
myHashSet.contains(3);               //返回 false(未找到)
myHashSet.add(2);                    //set=[1, 2]
myHashSet.contains(2);               //返回 true
myHashSet.remove(2);                 //set=[1]
myHashSet.contains(2);               //返回 false(已移除)
```

【限制】 $0 \leqslant key \leqslant 10^6$，最多调用 10^4 次 add、remove 和 contains。

【解题思路】 开放定址法(线性探测法)。相关原理参见 5.1.1 节，这里最多调用 10^4 次 add、remove 和 contains，所以设置哈希集合的长度为 MAXH=10 000，设置哈希函数为 $h(k)=k\%P$，P=9997，用 NULLKEY=-1 表示空位置键，DELKEY=-2 表示已删除位置的键。

(1) add(key)：求出 d=key%P，若 ht[d]=d 或者 ht[d]为空位置，或者 ht[d]为已删除位置，在 ht[d]插入 key(ht[d]=d 时插入 key 保证哈希表中的键唯一)，否则说明有冲突，使用线性探测法，即置 d=(d+1)%MAXH，再进行试探。

C++：

```
1  void add(int key) {                    //插入 key
2      int d=key%P;                       //求 d0
3      while(ht[d]!=key && ht[d]!=NULLKEY && ht[d]!=DELKEY) {
4          d=(d+1)%MAXH;                  //找到 NULLKEY、DELKEY 的位置 d
5      }
6      ht[d]=key;
7  }
```

(2) contains(key)：求出 d=key%P，若 ht[d]=d(查找成功)或者 ht[d]为空位置(查找失败)，结束查找，否则使用线性探测法，即置 d=(d+1)%MAXH，再进行试探。对于找到的 d，若该位置为 NULLKEY，说明查找失败，即哈希表中不包含 key，返回 false；若该位置不为 NULLKEY(因为 key≥0，该位置不可能为 DELKEY)，说明哈希表中包含 key，返回 true。

C++：

```
1  bool contains(int key) {               //是否包含 key
2      return ht[hash(key)]!=NULLKEY;
3  }
```

(3) remove(key)：按照 contains(key)的查找过程找到 key 的位置 d，若该位置不是 NULLKEY，则置为 DELKEY，否则不修改。

C++：

```
1  void remove(int key) {                    //删除 key
2      int d=hash(key);
3      if(ht[d]!=NULLKEY)
4          ht[d]=DELKEY;
5  }
```

扫一扫

源程序

Q：上述题目中约定 $0 \leqslant key \leqslant 10^6$，初学者 Chen 直接用一个长度为 10^6+1 的布尔数组 ht 作为哈希表，先初始化所有元素为 false，当插入 key 时置 ht[key]=true，请问 Chen 的实现和上述实现相比有什么优缺点？

A：Chen 的实现算法简单，不需要解决冲突，所以速度更快，查找、插入和删除算法的时间复杂度均为 $O(1)$，但占用的空间较多，并不能体现哈希表的本质。哈希表是将一个大数据范围映射为一个较小的数据范围，必须解决冲突问题。

5.2.2 LeetCode706——设计哈希映射★

【问题描述】 不使用任何内建的哈希表库设计一个哈希映射，实现 MyHashMap 类。

(1) MyHashMap()：用空映射初始化对象。

(2) void put(int key,int value)：求出 d=key%MAXH，向 HashMap 插入一个键值对(key,value)，如果 key 已经存在于映射中，则更新其对应的值 value。

(3) int get(int key)：返回特定的 key 所映射的 value，如果映射中不包含 key 的映射，返回−1。

(4) void remove(int key)：如果映射中存在 key 的映射，则移除 key 和它所对应的 value。

示例：

```
MyHashMap myHashMap=new MyHashMap();
myHashMap.put(1, 1);              //myHashMap 现在为[[1,1]]
myHashMap.put(2, 2);              //myHashMap 现在为[[1,1], [2,2]]
myHashMap.get(1);                 //返回 1,myHashMap 现在为[[1,1], [2,2]]
myHashMap.get(3);                 //返回−1(未找到),myHashMap 现在为[[1,1], [2,2]]
myHashMap.put(2, 1);              //myHashMap 现在为[[1,1], [2,1]](更新已有的值)
myHashMap.get(2);                 //返回 1,myHashMap 现在为[[1,1], [2,1]]
myHashMap.remove(2);              //删除键为 2 的数据,myHashMap 现在为[[1,1]]
myHashMap.get(2);                 //返回−1(未找到),myHashMap 现在为[[1,1]]
```

【限制】 $0 \leqslant key \leqslant 10^6$，最多调用 10^4 次 put、get 和 remove。

【解题思路】 拉链法。相关原理参见 5.1.1 节，设计哈希函数为 $h(k)=k\%MAXH$，置 MAXH=997。在哈希表的结点类型 SNode 中包含关键字 key、值 val 和指向下一个结点的指针 next。哈希表数组为 ht[0..MAXH−1]，其中 ht[i]存放指向单链表 i 的首结点(单链表 i 由哈希地址为 i 的结点组成)。

(1) MyHashMap()：将 ht 的所有元素初始化为 NULL。

(2) put(key,value)：求出 d=key%P，在单链表 d 中查找 key 的结点，若找到了这样的

结点 p，置结点 p 的值为 value，否则新建结点 q 存放< key，value >，将其插入单链表 d 的头部。

(3) get(key)：求出 d＝key%P，在单链表 d 中查找 key 的结点，若找到了这样的结点 p，返回结点 p 的值，否则返回－1。

(4) remove(key)：求出 d＝key%P，在单链表 d 中查找 key 的结点，若找到了这样的结点 p，通过前驱结点 pre 删除结点 p，否则不修改。

说明： 5.2.1 节的 LeetCode705 使用开放定址法实现，也可以使用拉链法实现。同样，5.2.2 节的 LeetCode706 使用拉链法实现，也可以使用开放定址法实现。

扫一扫

源程序

5.3 哈希集合应用的算法设计

5.3.1 LeetCode349——两个数组的交集★

问题描述参见第 1 章中的 1.3.6 节。

【解题思路】 用哈希集合实现除重。由于输出结果中的每个元素一定是唯一的，所以先用 hset1 对 nums1 除重，用 hset2 对 nums2 除重，再对 hset1 和 hset2 求交集 ans。对应的算法如下。

C++：

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        unordered_set<int> hset1,hset2;
        for(int x:nums1) hset1.insert(x);
        for(int y:nums2) hset2.insert(y);
        return intersection(hset1,hset2);
    }
    vector<int> intersection(unordered_set<int>&hset1,unordered_set<int>&hset2) {
        if(hset1.size()>hset2.size()) {
            return intersection(hset2,hset1);
        }
        vector<int> ans;
        for(int x:hset1) {          //求交集 ans
            if(hset2.count(x)) ans.push_back(x);
        }
        return ans;
    }
};
```

提交运行：

```
结果:通过;时间:12ms;空间:10.5MB
```

Python：

```
class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        hset1=set(nums1)
        hset2=set(nums2)
        return list(hset1 & hset2)
```

提交运行：

```
结果：通过；时间：24ms；空间：15.1MB
```

5.3.2 LeetCode202——快乐数★

【问题描述】 编写一个算法来判断一个数 n 是否为快乐数。对于一个正整数，每一次将该数替换为其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。如果这个过程的结果为 1，那么这个数就是快乐数。如果 n 是快乐数，则返回 true，否则返回 false。

扫一扫

源程序

例如，$n=19$，$1^2+9^2=82$，$8^2+2^2=68$，$6^2+8^2=100$，$1^2+0^2+0^2=1$，返回 true。

【限制】 $1 \leqslant n \leqslant 2^{31}-1$。

【解题思路】 用哈希集合实现除重。对于 n，按题目要求产生一个整数序列，当遇到 1 时表示 n 为快乐数。但是这个序列中可能出现相同的整数，如果这样，一定会陷入死循环，例如 $n=1$，$1^2=1$ 就是如此，所以需要除重，这里用哈希集合 hset 实现除重，也就是当 hset 中不包含 n 时才将 n 插入 hset 中。

5.3.3 LeetCode217——存在重复元素★

【问题描述】 给定一个整数数组 nums，如果任一数值在数组中至少出现两次，返回 true；如果数组中的每个元素互不相同，返回 false。

扫一扫

源程序

例如，nums=[1,2,3,1]，答案为 true；nums=[1,2,3,4]，答案为 false。

【限制】 $1 \leqslant$ nums.length $\leqslant 10^5$，$-10^9 \leqslant$ nums[i] $\leqslant 10^9$。

【解题思路】 用哈希集合实现判重。题目就是判断 nums 中是否存在重复的元素，若存在，返回 true，否则返回 false。这里用哈希集合 hset 实现判重，也就是用 x 遍历 nums，若 x 在 hset 中，返回 true，否则将 x 插入 hset 中。最后返回 false。

5.3.4 LeetCode379——电话目录管理系统★★

【问题描述】 设计一个电话目录管理系统，让它支持以下功能。

(1) PhoneDirectory(n)：初始化 0～$n-1$ 的 n 个电话号码。

(2) get()：给用户分配一个未被使用的电话号码，若获取失败返回−1。

(3) check(number)：检查电话号码 number 是否被使用。

(4) release(number)：释放电话号码 number，使其能够重新被分配。

示例：

```
PhoneDirectory directory=new PhoneDirectory(3);
//初始化电话目录，它包括 3 个电话号码，即 0、1 和 2
directory.get();
//可以返回任意未被分配的号码，这里假设返回 0
directory.get();
//假设函数返回 1
directory.check(2);
//号码 2 未被分配，所以返回 true
directory.get();
//返回 2，分配后只剩下一个号码未被分配
```

```
directory.check(2);
//此时号码 2 已经被分配,所以返回 false
directory.release(2);
//释放号码 2,将该号码变回未被分配状态
directory.check(2);
//号码 2 现在是未被分配状态,所以返回 true
```

【限制】 $1 \leqslant \text{maxNumbers} \leqslant 10^4$，$0 \leqslant \text{number} < \text{maxNumbers}$，调用方法的总数在[0，20000]范围内。

【解题思路】 用哈希集合实现判重。用一个队列 qu 存放所有未被分配的电话号码，用哈希集合 hset 存放已被分配的电话号码。

扫一扫

源程序

(1) PhoneDirectory(n)：将 0～$n-1$ 的 n 个电话号码进队。

(2) get()：若队列不为空，出队一个电话号码 x，并将其插入 hset 中，返回 x；否则说明没有未被分配的电话号码，返回−1。

(3) check(number)：判断 number 是否在 hmap 中。

(4) release(number)：从 hset 中删除 number，将其进队。

5.3.5 LeetCode128——最长连续序列★★

【问题描述】 给定一个未排序的整数数组 nums，找出数字连续的最长序列(不要求序列的元素在原数组中连续)的长度，设计并实现时间复杂度为 $O(n)$ 的算法解决此问题。

例如，nums=[100,4,200,1,3,2]，其中数字连续的最长序列是[1,2,3,4]，它的长度为 4，答案为 4。

【限制】 $0 \leqslant \text{nums.length} \leqslant 10^5$，$-10^9 \leqslant \text{nums}[i] \leqslant 10^9$。

【解题思路】 用哈希集合实现快速查找。用 ans 存放答案(初始为 0)。对于 nums 中的整数 x，将 $x-1$ 称为 x 的连续前驱，$x+1$ 称为 x 的连续后继，如果在 nums 中每个整数向后查找最大连续后继的个数，将最大值存放在 ans 中，最后返回 ans，这样的穷举算法的时间复杂度为 $O(n^2)$。

如果 x 存在最大连续整数序列 $x, x+1, \cdots, y$，则其连续序列的长度为 $y-x+1$，实际上从 $x+1$ 到 y 求出的连续序列的长度一定小于 $y-x+1$，所以仅求 x(即不存在连续前驱)的连续序列的长度即可。另外，要去掉重复的整数，并且需要快速判断其中是否存在某个整数，为此用哈希集合 hset 实现。先置 ans=0，求解过程如下：

(1) 将 nums 数组中的全部整数存放到哈希集合 hset 中。

(2) 用 x 遍历 hset，若 x 没有连续前驱，则求出其最长的连续后继，对应的连续序列的长度为 $y-x+1$，将最大值存放到 ans 中。

最后返回 ans。

例如，nums=[100,4,200,1,3,2,1,3,4]，将全部整数插入 hset 中，除重后 hset=[3,2,200,4,1,100]，ans=0，用 x 遍历 hset。

(1) $x=3$，存在 2，即 x 存在连续前驱，跳过。

(2) $x=2$，hset 中存在 1，即 x 存在连续前驱，跳过。

(3) $x=200$，hset 中不存在 199，置 $y=200$，在 hset 中查找 y 的最多连续后继，最终结果 $y=200$，ans=max(0，$y-x+1$)=1。

(4) $x=4$,hset 中存在 3,即 x 存在连续前驱,跳过。

(5) $x=1$,hset 中不存在 0,置 $y=1$,在 hset 中查找 y 的最多连续后继,依次为 $y=2$,$y=3$,$y=4$,最终结果 $y=4$,ans=max(0,$y-x+1$)=4。

(6) $x=100$,hset 中不存在 99,置 $y=100$,在 hset 中查找 y 的最多连续后继,最终结果 $y=100$,ans=max(4,$y-x+1$)=4。

hset 遍历后的答案为 ans=4。对应的算法如下。

C++:

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> hset;
        for(int x:nums) hset.insert(x);              //将全部整数存放到 hset 中
        int ans=0;                                   //存放答案
        for(int x:hset) {
            if(hset.find(x-1)==hset.end()) {         //若 x 不存在连续前驱
                int y=x;                             //以 x 为起点向后枚举
                while(hset.find(y+1)!=hset.end()) y++;
                ans=max(ans,y-x+1);                  //求最大长度
            }
        }
        return ans;
    }
};
```

提交运行:

```
结果:通过;时间:96ms;空间:48.5MB
```

上述算法尽管包含两重循环,但时间复杂度并不是 $O(n^2)$,实际上,第 7 行~第 13 行循环的执行次数最多为 $2n$(例如,nums=[1,2,…,n],遇到 $x=1$ 时第 10 行的 while 循环执行 n 次,而 x 为其他时仅执行一次,结果 ans=n),所以算法的时间复杂度为 $O(n)$。

Python:

```
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        hset=set(nums)                               #将全部整数存放到 hset 中
        ans=0                                        #存放答案
        for x in hset:
            if x-1 not in hset:                      #若 x 不存在连续前驱
                y=x                                  #以 x 为起点向后枚举
                while y+1 in hset: y+=1
                ans=max(ans,y-x+1)                   #求最大长度
        return ans
```

提交运行:

```
结果:通过;时间:72ms;空间:30.5MB
```

5.3.6 LeetCode41——缺失的第一个正数★★★

【问题描述】 给定一个未排序的整数数组 nums,找出其中没有出现的最小的正整数,请实现时间复杂度为 $O(n)$并且只使用常数级别额外空间的解决方案。

例如,nums=[3,4,-1,1],答案为 2; nums=[7,8,9,11,12],答案为 1。

【限制】 $1 \leqslant \text{nums.length} \leqslant 5 \times 10^5$，$-2^{31} \leqslant \text{nums}[i] \leqslant 2^{31}-1$。

【解法 1】 用哈希集合实现快速查找。将 nums 中的所有整数插入哈希集合 hset 中，ans 从 1 到 $n+1$ 查找，因为缺失的正数从 1 开始，并且最大为 $n+1$（此时 nums 中不包含 $1 \sim n$ 的任何数），如果 ans 包含在 hset 中，递增 ans 继续查找，否则说明 ans 就是缺失的第一个正数，返回 ans 即可。对应的算法如下。

C++：

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        unordered_set<int> hset;
        for(int x:nums) hset.insert(x);
        int ans=1;
        while(ans<=nums.size()+1) {
            if(hset.find(ans)==hset.end()) break;
            ans++;
        }
        return ans;
    }
};
```

提交运行：

```
结果:通过;时间:64ms;空间:45.4MB
```

Python：

```
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        hset=set()
        for x in nums: hset.add(x)
        ans=1
        while ans<=len(nums)+1:
            if ans not in hset: break;
            ans+=1
        return ans
```

提交运行：

```
结果:通过;时间:80ms;空间:28.5MB
```

【解法 2】 元素归位法。解法 1 的思路虽然简单，但是不符合空间复杂度为 $O(1)$ 的要求，改进算法的步骤如下：

（1）若 $\text{nums}[i]=i+1(0 \leqslant i \leqslant n-1)$，则 $\text{nums}[i]$ 称为归位元素，归位元素的取值范围是 $1 \sim n$。将所有能够归位且尚未归位的元素 $\text{nums}[i](1 \leqslant \text{nums}[i] \leqslant n$ 且 $\text{nums}[i] \neq \text{nums}[\text{nums}[i]-1])$ 通过交换操作（交换 $\text{nums}[i]$ 和 $\text{nums}[\text{nums}[i]-1]$）变为归位元素（$\text{nums}[i]$ 放在 $\text{nums}[\text{nums}[i]-1]$ 中），忽略其他元素。

（2）从 $i=0$ 开始查找没有归位元素的位置，即满足 $\text{nums}[i] \neq i+1$，则 $i+1$ 就是缺失的第一个正数，若没有找到这样的 i，则缺失的第一个正数为 $n+1$。

例如，$\text{nums}=[3,4,-1,1]$，$n=4$，归位元素的取值范围是 $1 \sim 4$，求解过程如下。

（1）$\text{nums}[0]=3$，3 是归位元素（其归位位置应该是 $\text{nums}[3-1=2]$）且尚未归位，交换 $\text{nums}[0]$ 和 $\text{nums}[2]$，$\text{nums}=[-1,4,3,1]$，此时 $\text{nums}[0]=-1$，不是归位元素。

（2）$\text{nums}[1]=4$，4 是归位元素且尚未归位，交换 $\text{nums}[1]$ 和 $\text{nums}[4-1=3]$，$\text{nums}=$

[−1,1,3,4]。nums[1]=1,1 是归位元素且尚未归位,交换 nums[1]和 nums[1−1=0],nums=[1,−1,3,4]。此时 nums[1]=−1,不是归位元素。

(3) nums[2]=3,3 是已归位元素。

(4) nums[3]=4,4 是已归位元素。

最后的归位结果如图 5.4 所示,nums[0]是归位元素,nums[1]是没有归位的元素,则答案为 2。

序号	0	1	2	3	⇨	0	**1**	2	3
元素	3	4	−1	1		1	**−1**	3	4

图 5.4 nums 的归位结果

对应的算法如下。

C++:

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        int n=nums.size();
        for(int i=0;i<n;i++) {
            while(nums[i]>=1 && nums[i]<=n && nums[nums[i]-1]!=nums[i]) {
                swap(nums[i],nums[nums[i]-1]);
            }
        }
        for(int i=0;i<n;i++) {
            if(nums[i]!=i+1) return i+1;
        }
        return n+1;
    }
};
```

提交运行:

```
结果:通过;时间:40ms;空间:30.5MB
```

Python:

```
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        n=len(nums)
        for i in range(0,n):
            while nums[i]>=1 and nums[i]<=n and nums[nums[i]-1]!=nums[i]:
                self.swap(nums,i,nums[i]-1)
        for i in range(0,n):
            if nums[i]!=i+1: return i+1
        return n+1

    def swap(self,nums,i,j):        #交换 nums[i]和 nums[j]
        nums[i],nums[j]=nums[j],nums[i]
```

提交运行:

```
结果:通过;时间:104ms;空间:24.3MB
```

5.3.7 LeetCode1436——旅行终点站★

【问题描述】 给定一份旅游路线图,该路线图中的旅行路线用数组 paths 表示,其中

paths[i]=[cityAi,cityBi],表示该路线将会从 cityAi 直接前往 cityBi。请找出这次旅行的终点站,即没有任何可以通往其他城市的路线的城市。题目数据保证路线图会形成一条不存在循环的路线,因此恰有一个旅行终点站。

例如,paths=[["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]],从"London"出发,最后抵达终点站"Sao Paulo",本次旅行的路线是"London"→"New York"→"Lima"→"Sao Paulo",答案为"Sao Paulo"。

【限制】 1≤paths.length≤100,paths[i].length=2,1≤cityAi.length,cityBi.length≤10,cityAi≠cityBi,所有字符串均由大小写英文字母和空格字符组成。

【解题思路】 用哈希集合实现快速查找。将所有的路线[a,b]看成二元组,由 a 构成全部前驱城市集合,旅行终点站一定为某个 b 并且它没有前驱,为此 A 用哈希集合 hset 表示。对应的算法如下。

C++:

```
class Solution {
public:
    string destCity(vector<vector<string>>& paths) {
        unordered_set<string> hset;
        for(auto x:paths) hset.insert(x[0]);
        for(auto x:paths) {
            if(hset.count(x[1])==0) return x[1];
        }
        return "";
    }
};
```

提交运行:

```
结果:通过;时间:12ms;空间:11.2MB
```

Python:

```
class Solution:
    def destCity(self, paths: List[List[str]]) -> str:
        hset=set()
        for x in paths:
            hset.add(x[0])
        for x in paths:
            if x[1] not in hset: return x[1]
```

提交运行:

```
结果:通过;时间:44ms;空间:15.1MB
```

5.4 哈希映射应用的算法设计

5.4.1 LeetCode350——两个数组的交集Ⅱ★

【问题描述】 给定两个整数数组 nums1 和 nums2,请以数组形式返回两个数组的交集,返回结果中每个元素出现的次数应该与元素在两个数组中都出现的次数一致(如果出现的次数不一致,则考虑取较小值),可以不考虑输出结果的顺序。

扫一扫

源程序

例如，nums1=[1,2,2,1]，nums2=[2,2]，答案为[2,2]。

【限制】 1≤nums1.length，nums2.length≤1000，0≤nums1[i]，nums2[i]≤1000。

【解题思路】 用哈希映射实现计数。题目的关键是每个数组中可能有重复的元素，这样交集中也可能存在重复的元素。用 ans 存放答案(初始为空)，保证 nums1 的长度较小，设计一个哈希表 hmap 累计 nums1 中每个元素出现的次数。用 y 遍历 nums2，当 hmap[y]>0 时说明 y 是交集元素，将其添加到 ans 中，同时递减 hmap[y]；否则说明 y 不是交集元素。最后返回 ans 即可。

5.4.2 LeetCode1460——通过翻转子数组使两个数组相等★

【问题描述】 给定两个长度相同的整数数组 target 和 arr，通过翻转子数组使两个数组相等。在每一步中可以选择 arr 的任意非空子数组并且将它翻转，能够执行此过程任意次。如果能让 arr 变得与 target 相同，返回 true，否则返回 false。

例如，target=[1,2,3,4]，arr=[2,4,1,3]，可以按照以下步骤使 arr 变成 target：

(1) 翻转子数组[2,4,1]，arr 变成[1,4,2,3]。

(2) 翻转子数组[4,2]，arr 变成[1,2,4,3]。

(3) 翻转子数组[4,3]，arr 变成[1,2,3,4]。

答案为 true。上述方法并不是唯一的，还存在多种将 arr 变成 target 的方法。

【限制】 target.length=arr.length，1≤target.length≤1000，1≤target[i]≤1000，1≤arr[i]≤1000。

【解题思路】 用哈希映射实现计数。依题意，只要 target 和 arr 两个数组的所有不同的整数序列相同并且每个整数出现的次数相同，则一定能通过翻转子数组使两个数组相等，否则不能达到目的。对应的算法如下。

C++：

```
class Solution {
public:
    bool canBeEqual(vector<int>& target, vector<int>& arr) {
        unordered_map<int,int> hmap;
        for(int x:target) hmap[x]++;
        for(int y:arr) {
            if(--hmap[y]<0) return false;
        }
        return true;
    }
};
```

提交运行：

```
结果:通过;时间:8ms;空间:15.4MB
```

Python：

```
class Solution:
    def canBeEqual(self, target: List[int], arr: List[int]) -> bool:
        hmap=dict()
        for x in target:                                    #计数
            if x in hmap:hmap[x]+=1
            else: hmap[x]=1
        for y in arr:
            if y in hmap:
```

```
                    if hmap[y]-1<0: return False
                    else:hmap[y]-=1
                else:return False
            return True
```

提交运行：

```
结果:通过;时间:40ms;空间:15.1MB
```

5.4.3 LeetCode383——赎金信★

【问题描述】 给定两个字符串 s 和 t，判断 s 能否由 t 中的字符组成，如果能，返回 true，否则返回 false。t 中的每个字符只能在 s 中使用一次。

例如，s="aa"，t="ab"，答案为 false；s="aa"，t="aab"，答案为 true。

【限制】 $1 \leqslant s.length, t.length \leqslant 10^5$，s 和 t 由小写英文字母组成。

【解题思路】 用哈希映射实现计数。设计一个哈希映射 hmap，先遍历 t 累计每种字母出现的次数，再遍历 s 递减相同字母的次数。若 hmap 出现值小于 0 的元素，则返回 false，否则返回 true。对应的算法如下。

C++：

```
class Solution {
public:
    bool canConstruct(string s,string t) {
        if(s.size()>t.size()) return false;
        unordered_map<char,int> hmap;
        for(char x:t) hmap[x]++;
        for(char y:s) hmap[y]--;
        for(auto z:hmap) {
            if(z.second<0) return false;
        }
        return true;
    }
};
```

提交运行：

```
结果:通过;时间:16 ms;空间:8.6MB
```

Python：

```
class Solution:
    def canConstruct(self, s: str, t: str) -> bool:
        if len(s)>len(t): return False
        hmap={}
        for x in t:
            if x in hmap: hmap[x]+=1
            else: hmap[x]=1
        for y in s:
            if y in hmap:hmap[y]-=1
            else:return False      #s 中的字符 y 在 t 中没有出现，返回 False
        for z in hmap.keys():
            if hmap[z]<0:return False
        return True
```

提交运行：

```
结果:通过;时间:64ms;空间:15.2MB
```

5.4.4 LeetCode347——前 k 个高频元素★★

【问题描述】 给定一个整数数组 nums 和一个整数 k，请返回其中出现频率前 k 高的元素，可以按任意顺序返回答案。

例如，nums=[1,1,1,2,2,3]，$k=2$，答案为[1,2]。

扫一扫

源程序

【限制】 $1 \leqslant$nums.length$\leqslant 10^5$，k 的取值范围是[1，数组中不相同的元素的个数]，题目数据保证答案唯一，换句话说，数组中前 k 个高频元素的集合是唯一的。

【解题思路】 用哈希映射实现计数。设计一个哈希映射计数器 hmap，遍历 nums 求出每个整数出现的次数。再设计一个按出现次数越小越优先的小根堆 minpq，遍历 hmap 将前 k 个元素进堆，对于剩余的元素 x，若 x 出现的次数大于堆顶元素出现的次数，则出堆一次，同时将 x 进堆。最后堆中的 k 个元素就是出现频率前 k 高的元素。

5.4.5 LeetCode242——有效的字母异位词★

【问题描述】 给定两个字符串 s 和 t，编写一个函数来判断 t 是否为 s 的字母异位词。若 s 和 t 中每个字符出现的次数都相同，则称 s 和 t 互为字母异位词。

例如，s="anagram"，t="nagaram"，答案为 true；s="rat"，t="car"，答案为 false。

【限制】 $1 \leqslant$s.length，t.length$\leqslant 5 \times 10^4$。s 和 t 仅包含小写字母。

【解法 1】 用哈希映射实现计数。设计一个哈希映射 hmap，先遍历 s 累计每种字母出现的次数，再遍历 t 递减相同字母的次数。若 hmap 中的全部元素为 0，返回 true，否则返回 false。对应的算法如下。

C++：

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        if(s.size()!=t.size()) return false;
        unordered_map<char,int> hmap;
        for(char x:s) hmap[x]++;
        for(char y:t) hmap[y]--;
        for(auto z:hmap) {
            if(z.second!=0) return false;
        }
        return true;
    }
};
```

提交运行：

```
结果:通过;时间:4ms;空间:7.2MB
```

Python：

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s)!=len(t): return False
        hmap={}
        for x in s:
            if x in hmap: hmap[x]+=1
```

```
            else: hmap[x]=1
        for y in t:
            if y in hmap:hmap[y]--=1
        for z in hmap.keys():
            if hmap[z]!=0:return False
        return True
```

提交运行：

```
结果:通过;时间:52ms;空间:15.3MB
```

【解法 2】 排序判断法。若 s 和 t 互为字母异位词，则它们排序的结果一定是相同的。对应的算法如下。

C++：

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        if(s.size()!=t.size())
            return false;
        sort(s.begin(),s.end());
        sort(t.begin(),t.end());
        return s==t;
    }
};
```

提交运行：

```
结果:通过;时间:12ms;空间:7.2MB
```

Python：

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s)!=len(t): return False
        s,t=list(s),list(t)
        s.sort()
        t.sort()
        return s==t
```

提交运行：

```
结果:通过;时间:80ms;空间:16.2MB
```

5.4.6 LeetCode205——同构字符串★

【问题描述】 给定两个字符串 s 和 t，判断它们是否为同构字符串。如果 s 中的字符可以按某种映射关系替换得到 t，那么这两个字符串是同构字符串。每个出现的字符都应当映射到另一个字符，并且不改变字符的顺序。不同字符不能映射到同一个字符上，相同字符只能映射到同一个字符上，字符可以映射到自己本身。

例如，s="egg"，t="add"，答案为 true；s="foo"，t="bar"，答案为 false。

【限制】 $1 \leqslant s.length \leqslant 5 \times 10^4$，t.length = s.length，s 和 t 由任意有效的 ASCII 字符组成。

【解题思路】 用哈希映射实现字符映射关系。设计两个哈希映射 hmap1 和 hmap2，hmap1[x]=y 表示 s 中的 x 字符映射到 t 中的 y 字符，hmap1[y]=x 表示 t 中的 y 字符

映射到 s 中的 x 字符。用 i 从 0 开始遍历，置 $x=s[i]$，$y=t[i]$，若 x 和 y 均没有建立映射关系，则置 hmap1[x]=y 和 hmap2[y]=x，建立 x 和 y 之间的双向元素关系。若 x 或者 y 已经建立映射关系但映射关系错误，返回 false，否则说明映射关系正确，继续遍历。遍历位置返回 true。对应的算法如下。

C++：

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        if(s.size()!=t.size()) return false;
        unordered_map<char,int> hmap1;
        unordered_map<char,int> hmap2;
        for(int i=0;i<s.size();i++) {
            char x=s[i],y=t[i];
            if(hmap1.count(x)==0 && hmap2.count(y)==0) {
                hmap1[x]=y;
                hmap2[y]=x;
            }
            else if(hmap1.count(x)==1 && hmap1[x]!=y || hmap2.count(y)==1
            && hmap2[y]!=x)
                return false;                    //不匹配返回 false
            }
            return true;
        }
};
```

提交运行：

```
结果:通过;时间:8ms;空间:7MB
```

Python：

```
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        if len(s)!=len(t): return False
        hmap1,hmap2={},{}
        for i in range(0,len(s)):
            x,y=s[i],t[i]
            if x not in hmap1 and y not in hmap2:
                hmap1[x],hmap2[y]=y,x
            elif(x in hmap1 and hmap1[x]!=y) or (y in hmap2 and hmap2[y]!=x):
                return False                    #不匹配返回 False
        return True
```

提交运行：

```
结果:通过;时间:36ms;空间:15.2MB
```

5.4.7 LeetCode1——两数之和★

【问题描述】 给定一个整数数组 nums 和一个整数目标值 target，请在该数组中找出和为目标值 target 的两个整数，并返回它们的数组下标。可以假设每种输入只会对应一个答案，但是数组中的同一个元素在答案中不能重复出现。可以按任意顺序返回答案。

例如，nums=[2,7,11,15]，target=9，答案为[0,1]；nums=[3,2,4]，target=6，答案为[1,2]。

【限制】 $2 \leqslant nums.length \leqslant 10^4$，$-10^9 \leqslant nums[i] \leqslant 10^9$，$-10^9 \leqslant target \leqslant 10^9$，只会存在一个有效答案。

【解题思路】 用哈希映射实现快速查找。设计一个哈希映射 hmap，其中元素 hmap[d]=i 表示 nums 数组中整数 d 的序号为 i。用 i 从 0 开始遍历 nums，若 target－nums[i]在 hmap 中，则两个元素 target－nums[i]（其序号为 hmap[target－nums[i]]）和 nums[i]的和为目标值 target，返回{hmap[target－nums[i]]，i}即可，否则置 hmap[nums[i]]=i。

扫一扫

源程序

5.4.8 LeetCode219——存在重复元素Ⅰ★

【问题描述】 给定一个整数数组 nums 和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j 满足 nums[i]=nums[j]且$|i-j| \leqslant k$，如果存在，返回 true，否则返回 false。

例如，nums=[1,2,3,1]，k=3，由于 nums1[0]=nums[3]，答案为 true；nums=[1,2,3,1,2,3]，k=2，答案为 false。

【限制】 $1 \leqslant nums.length \leqslant 10^5$，$-10^9 \leqslant nums[i] \leqslant 10^9$，$0 \leqslant k \leqslant 10^5$。

【解题思路】 用哈希映射实现快速查找。设计一个哈希映射 hmap，其中元素 hmap[d]=i 表示 nums 数组中整数 d 的序号为 i。用 i 从 0 开始遍历 nums，置 d=nums[i]，若 d 在 hmap 中并且满足条件 i－hmap[d]$\leqslant k$，返回 true，否则置 hmap[d]=i。遍历完毕返回 false。对应的算法如下。

C++：

```
class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        unordered_map<int,int> hmap;
        int n=nums.size();
        for(int i=0;i<n;i++) {
            int d=nums[i];
            if(hmap.count(d)==1 && i-hmap[d]<=k) return true;
            else hmap[d]=i;
        }
        return false;
    }
};
```

提交运行：

```
结果：通过；时间：128ms；空间：75.4 MB
```

Python：

```
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
        hmap={}                                    #定义一个字典
        for i,d in enumerate(nums):
            if d in hmap and i-hmap[d]<=k:         #找到后返回结果
                return True
            else:
                hmap[d]=i                          #添加到 hmap 中
        return False
```

提交运行：

```
结果：通过；时间：68ms；空间：26.1MB
```

5.4.9 LeetCode49——字母异位词的分组★★

【问题描述】 给定一个字符串数组 strs，请将字母异位词组合在一起，可以按任意顺序返回结果列表。字母异位词是通过重新排列源单词的字母得到的一个新单词，所有源单词中的字母通常恰好只用一次。

例如，strs=["eat","tea","tan","ate","nat","bat"]，答案为[["bat"],["nat","tan"],["ate","eat","tea"]]。

【限制】 $1 \leqslant strs.length \leqslant 10^4$，$0 \leqslant strs[i].length \leqslant 100$，$strs[i]$仅包含小写字母。

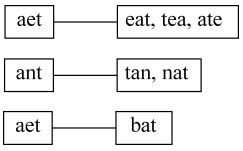


图 5.5 hmap 中存储的数据

【解题思路】 用哈希映射实现分组。从字符串角度看，显然所有字母异位词的排序结果是相同的，为此设计 unordered_map<string,vector<string>>类型的哈希映射 hmap，参考 5.4.5 节中的解法 2，以排序结果为键，以字母异位词为值，通过遍历 strs 产生 hmap，例如样例对应的 hmap 如图 5.5 所示，再遍历 hmap 将所有值存放到 ans 中，最后返回 ans 即可。

对应的算法如下。

C++：

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string,vector<string>> hmap;
        for(string x:strs) {
            string key=x;
            sort(key.begin(),key.end());
            hmap[key].push_back(x);
        }
        vector<vector<string>> ans;        //存放答案
        for(auto x:hmap) {
            ans.push_back(x.second);
        }
        return ans;
    }
};
```

提交运行：

```
结果:通过;时间:36ms;空间:20.1MB
```

Python：

```
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        hmap=dict()
        for x in strs:
            key="".join((lambda x:(x.sort(),x)[1])(list(x)))
            if key in hmap: hmap[key].append(x)
            else: hmap[key]=[x]
        ans=[]                                  #存放答案
        for v in hmap.values():
            ans.append(v)
        return ans
```

提交运行：

结果:通过;时间:56ms;空间:18.1MB

5.4.10 LeetCode249——移位字符串的分组★★

【问题描述】 给定一个字符串,对该字符串可以进行“移位”操作,也就是将字符串中的每个字母都变为其在字母表中后续的字母,如"abc"→"bcd",这样可以持续进行移位操作,从而生成移位序列"abc"→"bcd"→…→"xyz"。给定一个仅包含小写字母的字符串的列表strings,将该列表中所有满足移位操作规律的组合进行分组并返回。

例如,strings=["abc","bcd","acef","xyz","az","ba","a","z"],答案为[["abc","bcd","xyz"],["az","ba"],["acef"],["a","z"]]。可以认为字母表首尾相接,所以'z'的后续为'a',因此["az","ba"]也满足移位操作规律。

【解题思路】 用哈希映射实现分组。哈希映射 hmap 的设计与 5.4.9 节类似,其中键值为字符串的各字符转换为与第一个字符在字母表中的相对距离,将相同键值的字符串存放到对应的向量中。最后遍历 hmap 取出所有向量值即可。对应的算法如下。

C++:

```
class Solution {
public:
    vector<vector<string>> groupStrings(vector<string>& strings) {
        string key;
        unordered_map<string,vector<string>> hmap;
        for(string x:strings) {
            key="";
            for(int i=1;i<x.size();i++)
                key+=(x[i]-x[0]+26)%26;   //当前字符与首字符的距离
            hmap[key].push_back(x);
        }
        vector<vector<string>> ans;              //存放答案
        for(auto x:hmap)
            ans.push_back(x.second);
        return ans;
    }
};
```

提交运行:

结果:通过;时间:4ms;空间:7.9MB

Python:

```
class Solution:
    def groupStrings(self, strings: List[str]) -> List[List[str]]:
        hmap=dict()
        for x in strings:
            key=""
            for i in range(1,len(x)):
                key+=chr((ord(x[i])-ord(x[0])+26)%26)
                                                  #当前字符与首字符的距离
            if key in hmap: hmap[key].append(x)
            else: hmap[key]=[x]
        ans=[]                                    #存放答案
        for v in hmap.values():
            ans.append(v)
        return ans
```

提交运行：

结果：通过；时间：44ms；空间：15.1MB

推荐练习题

1. LeetCode3——无重复字符的最长子串★★
2. LeetCode136——只出现一次的数字★
3. LeetCode137——只出现一次的数字Ⅱ★★
4. LeetCode142——环形链表Ⅱ★★
5. LeetCode160——相交链表★
6. LeetCode268——丢失的数字★
7. LeetCode287——寻找重复数★★
8. LeetCode290——单词的规律★
9. LeetCode291——单词的规律Ⅱ★★
10. LeetCode451——根据字符出现的频率排序★★
11. LeetCode454——四数相加Ⅱ★★
12. LeetCode599——两个列表的最小索引总和★
13. LeetCode771——宝石与石头★
14. LeetCode804——唯一摩尔斯密码词★
15. LeetCode895——最大频率栈★★★
16. LeetCode1429——第一个唯一数字★★
17. LeetCode1399——统计最大组的数目★
18. LeetCode2347——最好的扑克手牌★