集合（set）

2019 年 3 月 8 日

1　集合（set）

集合一些元素的无序集群。集合中的元素可以是原子，也可以是集合，但集合中成员不能重复，且必须是同一类型。

多重集合，也称“包（bag）”，可以允许元素重复。

集合中元素在逻辑上没有次序，但在集合的实现过程中，为了方便比较或查找，需要为集合中元素规定顺序。集合可以保存实际数值，也可以保存指示信息，如在集合中保存性别信息：将男生保存为true，女生保存为false。

1.1　抽象数据类型接口（ADTI）

集合的基本操作包括并（union）、交（intersection）、差（difference）、判存在（contain）等。所以集合一般除构造、析构函数外，需要提供增加成员、删除成员函数；交、并、差、判存在、判相等函数。

```
bool AddMember(const T & elem); // 添加元素
bool DeleteMember(const T & elem); // 删除元素

bool Contains(const T & elem);  // 判包含
Set<T> & Union(Set<T> & a, Set<T> & b); // 求并集
Set<T> & Intersection(Set<T> & a, Set<T> & b); // 求交集
Set<T> & Difference(Set<T> & a, Set<T> & b); // 求差集
```

1.2 数据类型的实现：位向量

如果集合中保存元素代表的是某变量是否在集合之中，则可以用二进位数组表示。这样集合的并、交、差、判存在等操作将较为简单，使用按位与、交等操作即可。

如上所述，当两个对两个集合进行操作时，两个集合中元素必须处于对应位置。

1.3 数据类型的实现：有序链表

集合中元素在逻辑上是无序的，但是为了提高搜索效率，这里采用有序链表表示集合，顺序是升序。

求交集 因为集合本身是按一定顺序存储，因此求并集操作只需依次比较两个集合中元素即可：1）如果两个元素相等则插入到新集合；2）如果某个集合中对应元素较小，则对该集合的下一个元素进行比较。直至某集合中元素全部比较完毕，新集合中的元素就是这两个集合的交集。

求并集 并集就是两个集合相加后并保证集合中没有重复的元素。也可以通过依次比较集合中元素实现：1）如果两个元素相等，则插入到新集合中，并比较两个集合的下一个元素；2）如果其中一个元素较小，则将该元素插入到新集合中，并比较该集合的下一个元素；3）如果其中一个集合比较完毕，将另一个集合中未比较的元素，全部插入到新集合中。

求差集 依次比较两个集合中的元素：1）如果a中元素比较小，将该元素插入到新集合中，并比较a中下一个元素；2）如果b中元素小，也类似1）中操作；3）如果两个元素相等，则对比两个集合中的下一个元素；4）直至某个集合中元素全部比较完毕，将另一个集合中剩余的元素全部插入到新集合中。

判包含 如要判定集合中是否包含某元素x，可以利用集合中元素有序排列的特点，将指针快速指向比x小的最大一个元素，再比较其后一个元素是否与x相等。如果相等则集合中包含元素x；如果不相等，则集合中不包含该元素，并退出函数。不必依次比较每个元素。

2 并查集（disjoint set/union-find set）

并查集，将互不相交的集合按一定规律不断合并，合并之前需要查询并判断某元素应合并至哪个集合之中，这种集合称之为并查集，也称union-find set。

这种数据结构可以使用树来实现。其中合并操作可以通过将代表某集合的树的根结点，链接至代表另一集合的树上来实现。

3 字典（dictionary）

字典也是一种集合，但并不需要进行并、交、差等操作，只需要进行*判存在*操作，这类数据结构可以称之为字典（dictionary）。

字典中每个元素都附有一个称之为关键码、键（key）的域，每个元素的键均不相同。字典中元素至少含有两个域，一个存放键，一个存放其他数据。字典除插入、删除元素操作外，最重要的是实现*根据键，寻找对应元素的操作*。

字典可以根据具体需求选择线性表（数组和链表）的方式组织，还可以用二叉搜索树、多路搜索树的方式组织。这里以线性表和散列表为例。

3.1 数据类型的实现：线性表

为了保证搜索效率，使用线性表的方式实现字典需要保证线性表有序。具体实现方法可以参考集合部分使用有序链表实现的集合。

3.2 数据类型的实现：散列表（hash table）

与线性表的实现方式为了搜索效率，需要保证线性表有序。即使这样，搜索某元素时仍需多次比较。而散列表的实现方式是：*通过一个函数关系，将key和元素位置对应起来*，这样根据key值便可以直接访问某元素。这里的函数关系称为哈希函数（hash function）。

哈希表可以快速搜索和接近目标元素。

不过使用哈希函数时，有可能会出现冲突（collision），即某哈希函数将不同key值映射到了同一位置。称映射到同一位置的元素为同义词（synonym）。考虑到key值可以是任何值，而地址空间有限，这样多对少的映射难免会出现冲突。因此对哈希函数的选择提出了较高要求：1）哈希函数输入必须包含全部key值；2）当哈希表允许的长度为m时，哈希函数的输出应为[0,m-1]；3）哈希函数输出应均匀地分布在允许的地址空间中（以尽可能缓解冲突）。

3.2.1 哈希函数（hash function）

一般常采用的哈希函数为：

除留余数法（division）

$$hash(key) = key\%p$$

如果哈希表最长长度为m，则p为小于m的最大一个质数，并要求p不能接近2的幂。

数字分析法（digit analysis） 这种方法需要事先知道key值每一位的分布情况。

平方取中法（mid-square） 使散列长度为8的某次幂，比如r次幂，即

$$m = 8^r$$

再对内码[1]的平方取中间r 位作为散列地址。

折叠法（folding） 将key值自左到右分成多组，每组长度与散列地址空间位数相同。再经过诸如移位或分界等操作，获取散列地址。

3.2.2 处理冲突

冲突处理的基本思路是，将散列的m个地址，理解成m个桶，将地址有冲突的元素（即同义词）存放在同一个桶中，当桶满时，认为出现了溢出。

闭散列法（开地址法） 将散列数组设计成环形结构。当加入元素x时，出现冲突，则向后寻找最近一个空位置。

这样，当搜索某元素时，可以1）通过哈希函数算出地址，到该地址寻找该元素；2）如果对应位置上的元素不是待寻找元素，则查看下一个元素；3）如果下一个元素为空，则搜索失败；否则继续查看下一个元素；4）不断重复，直至a）寻找到对应元素或b）寻找到空位置或者回到原点。

[1]内码是一种二进制字符编码，类似ascii码。