# Chapter6 集合、字典

**扩充二叉树：要么含有叶节点，要么含有子节点数为2的节点**

## 6.1 集合及其表示

### 6.1.1 用位向量实现集合抽象数据类型

当集合是全集合 { 0, 1, 2, ..., n } 的一个子集，且 n 是不大的整数时，可用位(0, 1)向量来实现集合。

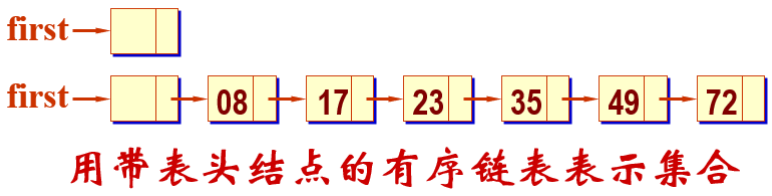
当全集合是由有限个可枚举的成员组成时，可建立全集合成员与整数 0, 1, 2, ...的一一对应关系，用位向量来表示该集合的子集。

一个二进位两个取值1或0，分别表示在集合与不在集合。如果采用16位无符号短整数数组bitVector[ ]作为集合的存储，就要考虑如何求出元素 i 在bitVector数组中的相应位置。

### 6.1.2 集合的位向量(bit Vector)类的定义

```
class bitSet {
//用位向量来存储集合元素，集合元素的范围在0到
//setSize-1之间。数组采用16位无符号短整数实现
private:
     int setSize;                //集合大小
     int vecterSize;             //位数组大小
     unsigned short *bitVector; //存储集合元素的位数组
}
```

### 6.1.3 用有序链表实现集合抽象数据类型




用带表头结点的有序链表表示集合

用有序链表来表示集合时，链表中的每个结点表示集合的一个成员。

各结点所表示的成员 $e_0$，$e_1$, ..., $e_n$ 在链表中按升序排列，即 $e_0 < e_1 < ... < e_n$。

集合成员可以无限增加。因此，用有序链表可以表示无穷全集合的子集。

## 6.2 等价类与并查集

### 6.2.1 等价关系与等价类

在求解实际应用问题时常会遇到等价类问题。

从数学上看，**等价类是对象（或成员）的集合，在此集合中所有对象应满足等价关系。**

若用符号 “°” 表示集合上的等价关系，则对于该集合中的任意对象x, y, z，下列性质成立：

自反性：x ° x (即等于自身)。

对称性：若 x ° y, 则 y ° x。

传递性：若 x ° y且 y ° z, 则 x ° z。

一个集合 S 中的所有对象可以通过等价关系划分为若干个互不相交的子集 $S_1$, $S_2$, $S_3$, ...，它们的并就是 S。**这些子集即为等价类。**

### 6.2.2 确定等价类的方法

1. 读入并存储所有的等价对( $i,j$ );
2. 标记和输出所有的等价类。

```
void equivalence ( ) {
   初始化;
    while ( 等价对未处理完 )
       { 读入下一个等价对 ( i,j );
         存储这个等价对 ; }
   输出初始化;
   for ( 尚未输出的每个对象 )
   输出包含这个对象的等价类 ;
}
```

### 6.2.3 并查集 (Union-Find Sets)

并查集支持以下三种操作：

Union (Root1, Root2)  //合并操作

Find (x)  //查找操作

UFSets (s)  //构造函数

对于并查集来说，每个集合用一棵树表示。

为此，采用树的双亲表示作为集合存储表示。集合元素的编号从0到 n-1。其中 n 是最大元素个数

**并查集的构造**

初始时, 用构造函数 UFSets(s) 构造一个森林, 每棵树只有一个结点, 表示集合中各元素自成一个子集合

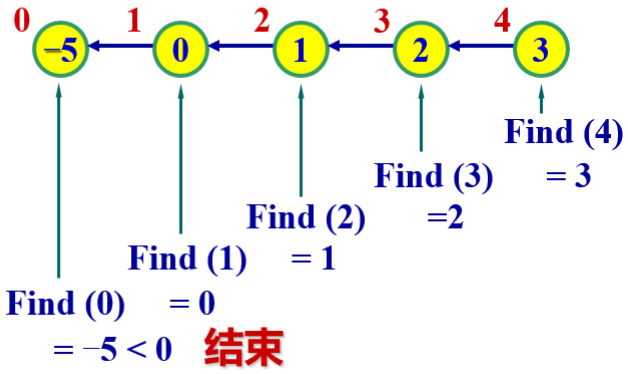
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| parent | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

用 Find(i) 寻找集合元素 i 的根。如果有两个集合元素 i 和 j, Find(i) == Find(j), 表明这两个元素在同一个集合中，

如果两个集合元素 i 和 j 不在同一个集合中,可用 Union(i, j) 将它们合并到一个集合中

**并查集的算法**

- **并查集操作的算法**
  - **查找**




执行一次Union操作所需时间是 O(1)，n-1次Union操作所需时间是O(n)。

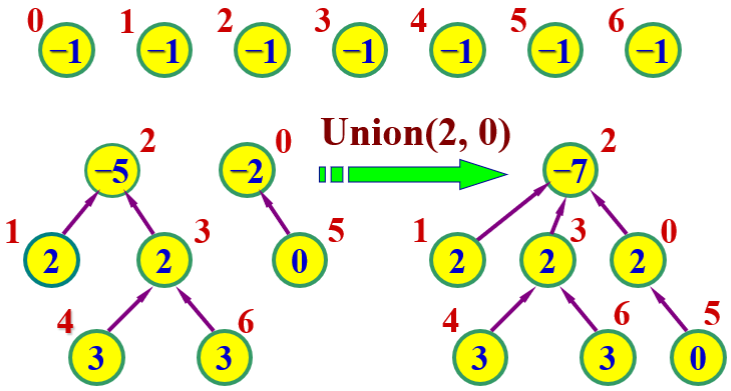
若再执行Find(0), Find(1), ..., Find(n-1), 若被搜索的元素为 i，完成 Find(i) 操作需要时间为O(i)，完成 n 次搜索需要的总时间将达到

$$O(\sum_{i=1}^{n} i) = O(n^2)$$

**改进的方法**

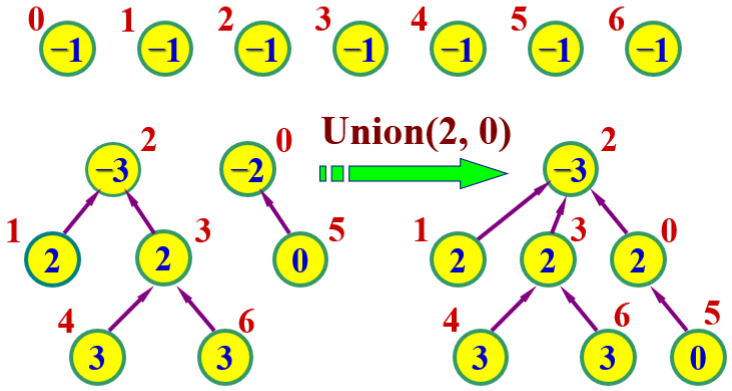
按树的结点个数合并

- **按树结点个数合并**
  - **结点个数多的树的根结点作根**




按树的高度合并

- **按树高度合并**
  - **高度高的树的根结点作根**




压缩元素的路径长度

## 6.3 字典（Dictionary）

### 6.3.1 字典定义与操作

字典是一些元素的集合，每个元素有一个称作关键码（key）的域，不同元素的关键码互不相同。

在讨论字典抽象数据类型时，把字典定义为<名字-属性>对的集合。

根据问题的不同，可以为名字和属性赋予不同的含义。

一般来说，有关字典的操作有如下几种：

1. 确定一个指定的名字是否在字典中；
2. 搜索出该名字的属性；
3. 修改该名字的属性；
4. 插入一个新的名字及其属性；
5. 删除一个名字及其属性。

### 6.3.2 字典的线性表描述

- **字典可以保存在线性序列 $(e_1,e_2,\ldots)$ 中，其中 $e_i$ 是字典中的元素，其关键码从左到右依次增大。为了适应这种描述方式，可以定义有序顺序表和有序链表。**
- **用有序链表来表示字典时，链表中的每个结点表示字典中的一个元素，各个结点按照结点中保存的数据值非递减链接，即$e_1 \le e_2 \le \ldots$。因此，在一个有序链表中寻找一个指定元素时，一般不用搜索整个链表。**

### 6.3.3 有序顺序表顺序搜索的时间代价

衡量一个搜索算法的时间效率的标准是：在搜索过程中关键码平均比较次数，也称为**平均搜索长度ASL (Average Search Length)**，通常它是字典中元素总数 n 的函数。

设搜索第 i 个元素的概率为 $p_i$，搜索到第 i 个元素所需比较次数为 $c_i$，则搜索成功的平均搜索长度:

$$ASL_{succ} = \sum_{i=1}^{n} p_i \cdot c_i. \quad (\sum_{i=1}^{n} p_i = 1)$$

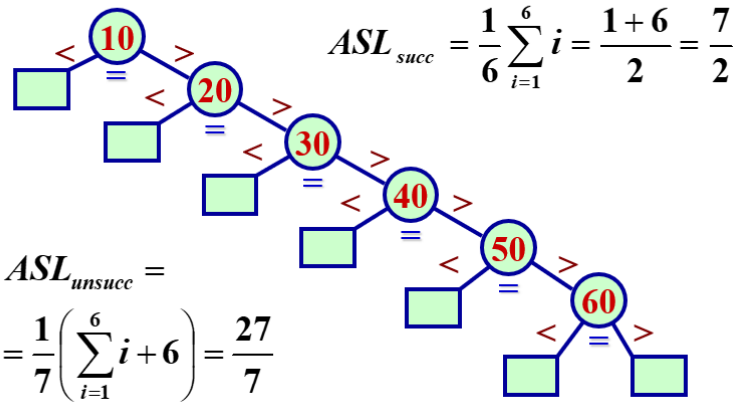
在顺序搜索情形，搜索第 i (1≤i≤n) 个元素需要比较 i 次，假定按等概率搜索各元素：

$$ASL_{succ} = \sum_{i=1}^{n} p_i \times c_i = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}\frac{(n+1)n}{2} = \frac{n+1}{2}$$

这与一般顺序表情形相同。但搜索不成功时不需一直比较到表尾，只要发现下一个元素的值比给定值大，就可断定搜索不成功。

设一个有 n 个表项的表，查找失败的位置有n+1个，可以用判定树加以描述。搜索成功时停在内结点，搜索失败时停在外结点。

- **例如，有序顺序表 (10, 20, 30, 40, 50, 60) 的顺序搜索的分析（使用判定树）**




#### 6.3.4 基于有序顺序表的折半搜索

为了加速搜索，在有序顺序表的情形，可以采用折半搜索，它也称二分搜索，时间代价可减到 $O(log_2 n)$

设 n 个元素存放在一个有序顺序表中。

折半搜索时, 先求位于搜索区间正中的元素的下标mid，用其关键码与给定值 x 比较:

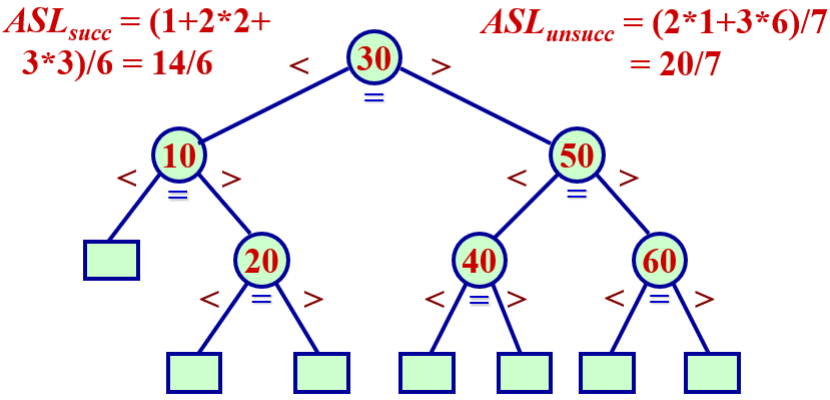
data[mid].key == x，搜索成功；

data[mid].key > x，把搜索区间缩小到表的前半部分，继续折半搜索；

data[mid].key < x，把搜索区间缩小到表的后半部分，继续折半搜索。

如果搜索区间已缩小到一个对象，仍未找到想要搜索的对象，则搜索失败

- **分析有序顺序表 ( 10, 20, 30, 40, 50, 60 ) 的折半搜索算法性能的判定树：**




判定树也是扩充二叉树，搜索成功时检测指针停留在树中某个内结点上。搜索不成功时检测指针停留在某个外结点（失败结点）上。

## 6.4 散列表（Hash Table）

### 6.4.1 散列表与散列函数

理想的搜索方法是可以不经过比较，一次直接从字典中得到要搜索的元素。

建立一个元素存储位置Address与其关键码key之间的对应函数关系Hash()：

Address = Hash(key)

在插入时依此函数计算存储位置并按此位置存放。**在搜索时对元素的关键码进行同样的计算，把求得的函数值当做元素存储位置， 在结构中按此位置搜索。这就是散列方法**

**按散列方法构造出来的表叫做散列表。所用转换函数叫做散列函数**

使用散列方法进行搜索不必进行多次关键码的比较, 搜索速度比较快, 可以直接到达或逼近具有此关键码的表项的实际存放地址。

### 6.4.2 同义词与其处理

对不同的关键码，通过散列函数的计算，可能得到了同一散列地址。

**称这些产生冲突的散列地址相同的不同关键码为同义词。**

由于关键码集合比地址集合大得多, 冲突很难避免。所以对于散列方法, 需要讨论以下两个问题：

1）对于给定的一个关键码集合，选择一个计算 简单且地址分布比较均匀的散列函数，避免或尽量减少冲突；

2）拟订解决冲突的方案。

### 6.4.3 散列函数

**构造散列函数时的几点要求**

散列函数应是简单的，能在较短的时间内 计算出结果。

散列函数的定义域必须包括需要存储的全部关键码，如果散列表允许有 m 个地址时，其值域必须在 0 到 m-1 之间

散列函数计算出来的地址应能均匀分布在整个地址空间中：若 key 是从关键码集合中随机抽取的一个关键码, 散列函数应能以同等概率取0 到 m-1 中的每一个值

**直接定址法**

此类函数取关键码的某个线性函数值作为散列地址：

Hash(key) = a*key+b {a, b为常数}

这类散列函数是一对一的映射，一般不会产生冲突。但它要求散列地址空间的大小与关键码集合的大小相同。

**数字分析法**

设有 n 个 d 位数，每一位可能有 r 种不同的符号。这 r 种不同符号在各位上出现的频率不一定相同。根据散列表的大小, 选取其中各种符号分布均匀的若干位作为散列地址。

n计算各位数字中符号分布均匀度的公式：

$$\lambda_k = \sum_{i=1}^{r} (\alpha_i^k - n/r)^2$$

其中， $a_i^k$表示第 i 个符号在第 k 位上出现的次数，n/r 表示各种符号在 n 个数中均匀出现的期望值。

**数字分析法仅适用于事先明确知道表中所有关键码每一位数值的分布情况，它完全依赖于关键码集合。如果换一个关键码集合，选择哪几位要重新决定**

**除留余数法**

设散列表中允许地址数为m，取一个不大于 m，但最接近于或等于 m 的质数 p 作为除数，用以下函数把关键码转换成散列地址：

**hash (key) = key % p p <= m**

其中，“%”是整数除法取余的运算，要求这时的质数 p 不是接近 2 的幂。

**平方取中法与折叠法**

**6.4.4 处理冲突的闭散列方法**

**因为任一种散列函数也不能避免产生冲突，因此选择好的解决冲突的方法十分重要。**

**线性探查法**

需要搜索或加入一个表项时, 使用散列函数计算关键字在表中的位置：

$H_0$ = hash ( key )

一旦发生冲突，在表中顺次向后寻找下一个位置 $H_i$ ：

$H_i$ = ( $H_{i-1}$ +1 ) % m， i =1, 2, ..., m-1

即用以下的线性探查序列在表中寻找下一 个位置：

$H_0$ +1, $H_0$ +2, ..., m-1, 0, 1, 2, ..., $H_0$-1

亦可写成如下的通项公式：

$H_i$ = ( $H_0$ + $i$ ) % m， i =1, 2, ..., m-1

**平方探查法**

$H_i$ = ( $H_0$ + $i^2$ ) % m， i =1, 2, ..., m-1

**二次探查法**

$H_i$ = ( $H_0$ + $i$ · hash_2(x) ) % m， i =1, 2, ..., m-1

### 6.4.5 处理冲突的开散列方法（链地址法）

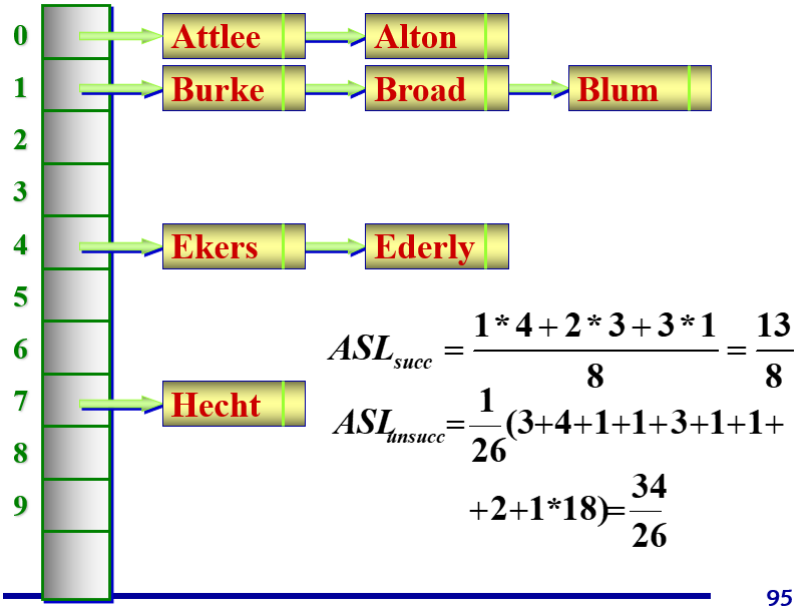
开散列方法首先对关键码集合用某一个散列函数计算它们的存放位置。

若设散列表地址空间的位置从 0 ~ m-1, 则关键码集合中的所有关键码被划分为 m 个子集，具有相同地址的关键码归于同一子集。我们称同一子集中的关键码互为同义词。每一个子集称为一个桶。

通常各个桶中的表项通过一个单链表链接起来，称之为同义词子表。

**所有桶号相同的表项都链接在同一个同义词子表中，各链表的表头结点组成一个向量。**

**向量的元素个数与桶数一致。桶号为i的同义词子表的表头结点是向量中第 i 个元素**




$$ASL_{succ} = \frac{1*4+2*3+3*1}{8} = \frac{13}{8}$$

$$ASL_{unsucc} = \frac{1}{26}(3+4+1+1+3+1+1+2+1*18) = \frac{34}{26}$$