第7章 散列表

学习要点:

- 理解集合的概念。
- 理解以集合为基础的抽象数据类型。
- 掌握用位向量实现集合的方法。
- 掌握用链表实现集合的方法。

返回章目录

7.1 以集合为基础的抽象数据类型

7.1.1 集合基础知识

- 集合是成员(对象或元素)的一个群集。集合中的成员可以是原子(单元素)，也可以是集合。

- 集合的成员必须互不相同。

- 在算法与数据结构中所遇到的集合，其单元素通常是整数、字符、字符串或指针，且同一集合中所有成员具有相同的数据类型。

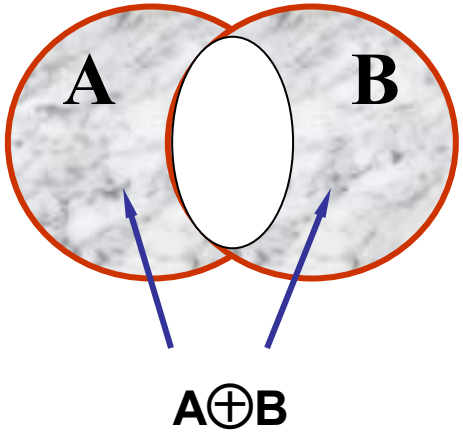
例： color = { red, orange, yellow, green,
black, blue, purple, white }

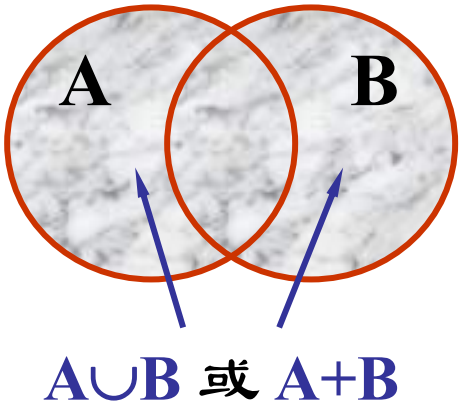
name = { “Li”, “Cao”, “Liu”, “Ma”,
“Peng”, “Wang”, “zhang” }

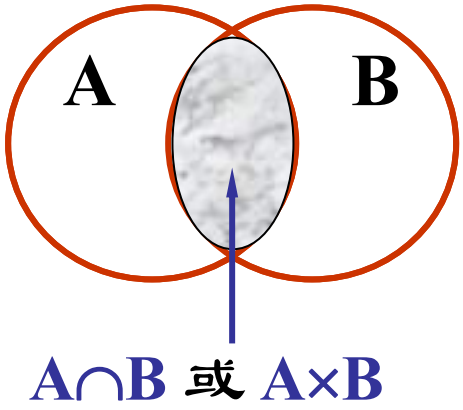
- 集合中的成员一般是无序的，但在表示它时，常写在一个序列里。
- 常设定集合中的单元素具有线性有序关系，此关系可记作“<”，表示“优先于”。
- 整数、字符和字符串都有一个自然线性顺序。指针也可依据其在序列中安排的位置给予一个线性顺序。

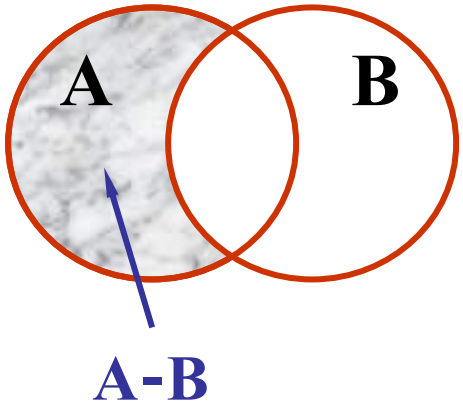
集合的基本运算:

- 集合之间的并：**A∪B**
- 集合之间的交：**A∩B**
- 集合之间的差：**A-B**
- 集合之间的异或：**A⊕B=(A-B) ∪(B-A)**









7.1 以集合为基础的抽象数据类型

7.1.2 ADT集合支持的基本运算

约定：其中大写字母表示一个集合，小写字母表示集合中的一个元素。

(1) SetUnion(A,B):并集运算。
(2) SetIntersection(A,B):交集运算。
(3) SetDifference(A,B):差集运算。
(4) SetAssign (A,B):赋值运算。
(5) SetEqual(A,B):判等运算。
(6) SetMember(x,S):成员运算。
(7) SetInsert(x,S):插入运算。
(8) SetDelete(x,S):删除运算。

返回章目录

7.2 用位向量实现ADT集合

约定所讨论的集合都是全集合{1, 2, …, N }的子集，而且N是一个不大的固定整数。对此，任何一个集合A⊆{ 1，2，…，N }，都可以用它的特征函数：

$$\delta_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$$

来表示。而特征函数$\delta_A(x)$可以用位向量来表示：

V[size-1]			……	V[3]	V[2]	V[1]	V[0]

7.2 用位向量实现ADT集合

用位向量实现的集合结构Bitset定义

```
typedef struct bitset * Set;
typedef struct bitset
{
        int  setsize;              //集合大小
        int  arraysize;            //位数组大小
        unsigned short *v;         //位数组 16位
}Bitset;
```

7.2 用位向量实现ADT集合

通过检测元素在表示集合的位向量中相应位实现成员函数**SetMember(x,S)**

```
int SetMember(int x, Set S)
{
    if (x < 0 || x >= S->setsize) Error("BadInput");
    return S->v[ArrayIndex(x)] & BitMask(x);
}
```

7.2 用位向量实现ADT集合

元素在数组中下标定位函数ArrayIndex和位屏蔽函数BitMask:

下标定位函数ArrayIndex通过将x右移4位获得x在数组中的位置。位屏蔽函数BitMask则先计算出x除以16的余数y，然后将1左移y位确定x在相应数组单元中的准确位置。

```
int ArrayIndex(int x)
{
    return int(x) >> 4;
}
unsigned short BitMask(int x)
{
    return 1 << (int(x) & 15);
}
```

7.2 用位向量实现ADT集合

通过集合A和B的位向量按位或来重载并集运算SetUnion(A, B)。

```
Set SetUnion (Set A, Set B)
{
    int i;
    Set tmp=SetInit(A->setsize);
    for (i = 0; i < A->arraysize; i++) tmp->v[i] = A->v[i] | B->v[i];
    return tmp;
}
```

返回章目录

7.3 用链表实现ADT集合

用有序链表实现集合Set如下：

```
typedef struct list * Set;
typedef struct list
{
    link  first;       //指向第一个元素的指针
}List;
```

```
Set SetIntersection (Set A, Set B)  //交集运算实现
{   link a, b, p, q, r;
    Set tmp = SetInit( );
    a= A->first; b= B->first; p= NewNode( );    q= p;
    while (a && b){
      if (a->data == b->data){
        r = NewNode( ); r->data = a->data;  r->next=0;
        p->next=r; p=r;
        a=a->next;  b=b->next; }
      else if (a->data < b->data) a=a->next;
      else b=b->next;          }
    if (p!=q) tmp->first=q->next;
    free (q);
    return tmp;
}
```

返回章目录

7.4 集合的应用
—Eratosthenes筛法

符号表

学习要点：

- 理解抽象数据类型符号表的概念。
- 掌握数组实现符号表的方法。
- 理解开散列和闭散列的概念。
- 掌握用开散列表实现符号表的方法。
- 掌握除余法、数乘法、平方取中法、基数转换法和随机数法等散列函数构造方法。
- 掌握采用线性重新散列技术的闭散列表实现符号表的方法。

返回章目录

7.5 实现符号表的简单方法

- ADT符号表的概念
  - 以集合为基础，并支持Member、Insert和Delete三种运算的抽象数据类型叫做符号表。
- ADT符号表的应用
  - 公司的客户符号表
  - 一个地区的固定/移动电话号码符号表
  - 软件开发中的数据符号表
  - 网上的在线符号表
  - 互联网/企业网/局域网网管中的IP地址符号表等等

7.5 实现符号表的简单方法

7.5.2 用固定数组实现符号表

7.5.2.1数组实现符号表类Adictionary的定义

```
typedef struct  atab  *Table;
Typedef struct atab{
        int arraysize;
        int last;
        setitem *data;
}Atab;
```

7.5 实现符号表的简单方法

7.5.2 用固定数组实现符号表

7.5.2.2 优缺点

- 优点：
  - 结构简单，实现操作的逻辑简单。
- 缺点：
  - 所表示的集合的大小受到数组大小的限制。
  - 三个基本操作在最坏情况下都需要O(n)。
  - 通常集合元素并不占满整个数组，空间没有得到充分利用。

返回章目录

7.6 用散列表实现符号表

7.6.1 实现符号表的简单方法——开散列

基本设想：

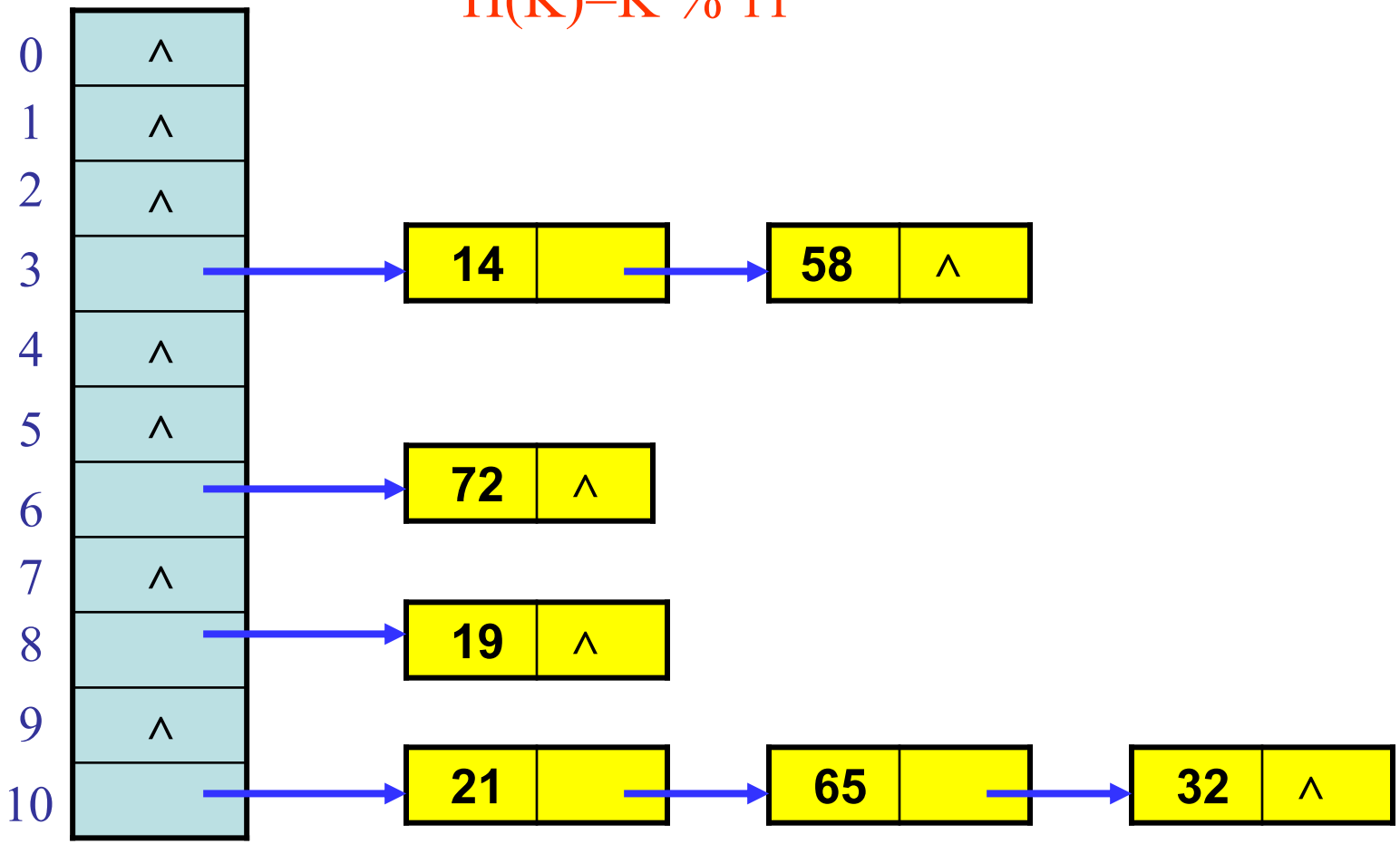
把符号表中的所有元素散列(hashing)即映射到一个桶数组(散列表)的桶中。当有多个不同元素被散列到同一个桶时，这些元素用链在一个表里。

期望散列能均匀，使得当桶数组的规模与符号表的规模同阶时，桶数组的每一个桶中大致有常数个元素，从而对符号表的三个基本操作都只需要常数时间。

这里的问题是如何散列即如何构造散列(映射)函数去达到设想的期望？

例　一组关键字为（21， 14， 19，58， 65， 32， 72）

H(K)=K % 11



7.6 用散列表实现符号表

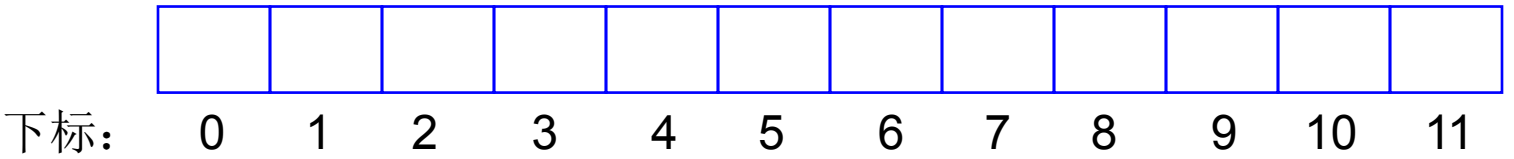
7.6.2 用闭散列实现符号表

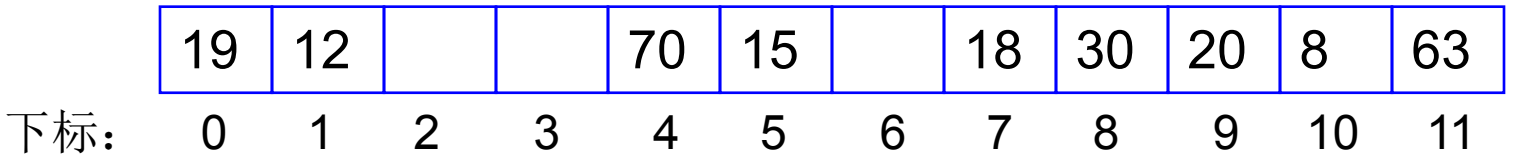
7.6.2.1 与开散列的相同点和区别

- 相同点：把符号表中的所有元素散列(hashing)即映射到一个桶数组(散列表)的桶中。
- 区别：桶数组(散列表)的桶直接用来存放符号表元素，而且一个桶只存放一个元素。出现多个元素散列到同一个桶时(这叫冲突)，需要按照确定的规则在桶数组中进行探测，找还有没有存放元素的桶(这叫空桶)，然后将发生冲突的元素放入空桶，解决冲突，实现散列。

例 一组关键字为（20，30，70，15，8，12，18，63，19）

$H(K)=K \% 11$





用线性探测法处理冲突！

7.6 用散列表实现符号表

7.6.2.2 散列方法需要解决的问题：

1、 构造好的散列函数

（1）所选函数尽可能简单，以便提高转换速度。

（2）所选函数对关键字计算出的地址，应在散列地址集中大致均匀分布，以减少空间浪费。

2、 制定解决冲突的方案。

3、需要对ht[ ]的每一个桶的状态加以标记：

state[k]=0表示ht[k]桶存放着元素。

state[k]=1表示ht[k]桶一直是空桶。

state[k]=2表示ht[k]桶现在是空桶但曾经存放过元素

7.6 用散列表实现符号表

7.6.2.3 散列函数的构造方法

1、直接定址法：取关键字的某个线性函数值作为散列地址。——较直观的方法

H(key)=a*key+b; //a、b为常数

2、除留余数法：取关键字的某个不大于表长m的数p除后所得的余数作为散列地址。——较简单、常用的方法

H(key)=key%p; //p<=m

3、平方取中法：取关键字平方后的中间几位作为散列地址(若超出范围时，可再取模)。

7.6 用散列表实现符号表

7.6.2.3 散列函数的构造方法

4、折叠法：将关键字分成几个部分，再将这几个部分结合(作某一运算)起来的值作为散列地址。——常用于关键字的位数较多的情况

5、数值分析法：如果事先知道所有可能的关键字的取值时，可通过对这些关键字分析，发现其变化规律，构造出相应的函数。

7.6 用散列表实现符号表

7.6.2.4 处理冲突的方法

1、线性探测法：当由H(k)算出的位置不空时，依次用下面函数以找出一个新的空位置。

$H_i(k)=(H(k)+d_i) \bmod m$,其中 $i=1,2,\ldots k(k\leq m-1)$

其中：m为散列表长度，d_i的取值常用如下形式：di=i，即di为增量序列，依次取值 1，2，……，m-1

7.6 用散列表实现符号表

7.6.2.4 处理冲突的方法

例：已知散列表的地址区间为0～11，散列函数为H(k)=k % 11,采用线性探测法处理冲突，试将以下关键字序列依次存储到散列表中，构造出该散列表，并求出在等概论情况下的平均查找长度。

20，30，70，15，8，12，18，63，19

H(20)=9，可直接存放到A[9]中(搜索时次数为1)；

……

H(19)=8， 因为下标为8~11的元素均已被占用，故往后搜索并绕回到A[0]存放(搜索时次数为5) 。

7.6 用散列表实现符号表

7.6.2.4 处理冲突的方法

下标：	0	1	2	3	4	5	6	7	8	9	10	11
	19	12			70	15		18	30	20	8	63
搜索次数：	5	1			1	2		1	1	1	3	4

在等概率情况下，该表成功的平均查找长度如下：

（1×5＋2×1＋3×1＋4×1＋5×1）/9＝19/9

7.6 用散列表实现符号表

7.6.2.4 处理冲突的方法

2、二次重新散列技术

它选取的探查桶序列为：$h(x), h_1(x), h_2(x), \ldots, h_{2i}(x), h_{2i+1}(x)$，

其中，$h_{2i}(x) = \left(h(x) - i^2\right)\%B$ ，$h_{2i-1}(x) = \left(h(x) + i^2\right)\%B$ 。

3、随机重新散列技术

它选取的探查桶序列为：$h_i(x) = \left(h(x) + d_i\right)\%B$,i=1，2，…，B-1。

其中，$d_1, d_2, \cdots, d_{B-1}$ 是1，2，…，B-1的一个随机排列。

4、双重散列技术

这种方法使用2个散列函数h和h’来产生探索序列：

$$h_i(x) = \left(h(x) + ih'(x)\right)\%B$$

其中 i=1，2，…，B-1。要求h′(x)的值和B互素 。

7.6 用散列表实现符号表

7.6.3 改进Delete(x)的基本思想：

动机是希望腾出的空桶(记为ht[i])不仅仅可供新元素插入，而且能为提高还在桶数组中的元素(比如y= ht[j])的查找速度作出贡献：减少查找y的探测次数。

容易理解，如果不作任何改进，查找y的探测次数会等于插入y时的探测次数。如果插入y时x已在桶ht[i]中且与x发生冲突而增加了插入的探测次数，那么，现在x不存在了，只要将x腾出的桶ht[i]让y顶替，就可以使得将来查找y时减少探测次数。因此改进Delete(x)的途径是在当前的桶数 组中找能顶替x的y。如果找不到，就按Delete(x)的原版处理；如果找得到，在用y顶替x之后还可以引起连锁反应。

7.6 用散列表实现符号表

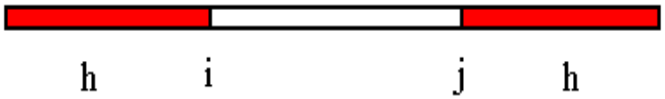
- 改进Delete(x)的细节—找能顶替x的y

假设被删除元素x位于桶单元ht[i]。现考察一个非空单元ht[j]中的元素y，其散列函数值设为h=hf(y)，则按从h出发的线性探测，只要i比j离h近即可使得在顶替后找y的探测次数减少。

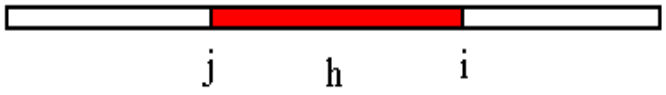
①当i<j时，若i<h≤j，则不可用元素ht[j]顶替ht[i]；若h≤i<j或i<j<h，则可用元素ht[j]顶替ht[i]。如下图(a)。

②当j<i时，若j<h≤I，则可用元素ht[j]顶替ht[i]，如下图 (b)；否则不可。

这里以线性探测为前题，以顶替后减少探测次数为目标。



(a)



(b)

7.6 用散列表实现符号表

7.6.4 用散列表实现符号表的效率

若能选择一个好的散列函数，将集合中的N个元素均匀地散列到B个桶中，那么，每个桶中平均有N/B个元素，使得在开散列表中，Insert，Delete和Member运算都只要O(N/B)的平均时间。进而当N/B为一常数时，符号表的每一个运算都可在常数时间内完成。

因此：对于开散列表，关键在于选择一个好的散列函数

返回章目录

7.7 符号表的应用

字符串频率统计问题

★问题描述：

在对文本文件进行处理时会遇到对特定字符串在文件中重复出现次数统计的问题。散列表方法可以用来统计文本文件中字符串出现的频率。

输入文件示例

input.txt

```
Columbus Washington Napoleon Washington Lee Grant
Washington Lincoln Grant Columbus Washington
```

输出文件示例

output.txt

```
Lee 1
Grant 2
Lincoln 1
Napoleon 1
Columbus 2
Washington 4
```

THE END