第8章 集合

- 以集合为基础的抽象数据类型
  - 集合的定义和记号
  - 定义在集合上的基本运算
- 用位向量实现集合
- 用链表实现集合
- 应用举例

8.1 以集合为基础的抽象数据类型

- 集合是表示事物的最有效的数学工具之一，生活中也随处可见集合的例子。
- 如：银行中所有储户帐号的集合；图书馆中所有藏书的集合；一个程序中所有标识符的集合；**2003**级**34**班全体同学的集合等 。
- 在数据结构和算法的设计中，集合是许多重要抽象数据类型的基础。

8.1.1 集合的定义

- 集合：集合是由元素(成员)组成的一个类。集合的成员可以是一个集合，也可以是一个原子。
- 原子：不可再分的元素。
- 多重集合：允许同一元素在集合中多次出现的集合。
- 有序集：当由原子组成的集合具有线性序关系“<”时，称该集合为有序集。“<”是集合的一个线性序，它满足：
  - (1)若a、b是集合中任意2个原子,则a<b,a=b和b<a三者必居其一；
  - (2)a，b和c是集合中的原子，且a<b，b<c则a<c(传递性)。

8.1.1 集合的定义

- 记录：集合中的元素通常是记录。
- 项(域)：元素的属性。元素通常有多个项。
- 键(值)：能唯一地标识元素的值。它也是元素的属性。有序集通常以键(值)的序为序。
- 为方便叙述，常将一个元素当作一个键来处理，但要记住键只是元素记录中许多域中的一个域。

8.1.2 集合的记号

- 枚举式：把集合的元素枚举在一对花括号中。

  例如｛**1**，**4**｝表示由**1**和**4**两个元素组成的集合。集合中元素的枚举顺序是任意的。例如｛**1**，**4**｝和｛**4**，**1**｝表示同一集合。

  这种方式只对有穷集可行。

- 解析式：把集合的元素应满足的条件解析地表达在一对花括号中。

  例如 ｛**x**｜存在整数**y**，使**x = y²**｝表示由全体完全平方数组成的集合。

  这种方式既可表示有穷集，又可表示无穷集。

8.1.3 定义在集合上的基本运算

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。

- **SetUnion(A,B)**：并集运算
- **SetIntersection(A,B)**：交集运算
- **SetDifference(A,B)**：差集运算
- **SetAssign(A,B)**：赋值运算(将B的值赋给A)
- **SetEqual(A,B)**：判等运算
- **SetMember(x,S)**：成员运算(x是否属于S)
- **SetInsert(x,S)**：插入运算
- **SetDelete(x,S)**：删除运算

8.2 用位向量实现集合

- 位向量是一种每个元素都是二进制位（即**0/1**值）的数组。
- 当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时，可以用位向量来实现集合。此时，对于任何一个集合**A⊆{1,2,…,N }**，可以定义它的特征函数为：

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

- 用一个**n**位的向量**v**来存储集合**A**的特征函数值**v[i]=** δ_A**(i)**，**i=1,2,…,n**，可以唯一地表示集合**A**。位向量**v**的第**i**位为**1**当且仅当**i**属于集合**A**。

8.2 用位向量实现集合

- 这种表示法的主要优点是**SetMember**，**SetInsert**和**SetDelete**运算都可以在常数时间内完成，只要访问相应的位即可。
- 在这种集合表示法下，执行并集运算、交集运算和差集运算所需的时间正比于全集合的大小**n**。

- 位向量实现的集合结构**BitSet**的定义

```
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
}Bitset;
```

- 创建一个用位向量实现，可存储集合大小为**size**的空集

```
Set SetInit(int size)
{
    int i;
    Set S=malloc(sizeof *S);
    S->setsize=size;
    S->arraysize=(size+15)>>4;
    S->v=malloc(arraysize*sizeof(unsigned short));
    for (i=0; i<arraysize; i++)   S->v[i]=0;
    return S;
 }
```

- 通过复制表示集合的位向量实现赋值运算

```
void SetAssign(Set A, Set B)
{
    int i;
    if (A->setsize !=B->setsize)
       Error("Sets are not the same size");
    for(i=0; i<A->arraysize; i++)
       A->v[i]=B->v[i];
 }
```

- 通过检测元素在表示集合的位向量中相应位，来判定成员属性

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

```
int ArrayIndex(int x)
{
   return x>>4;
 }
```

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

- 通过检测集合**A**和**B**的位向量来判定**A**和**B**是否相等

```
int SetEqual(Set A, Set B)
{
   int i,retval=1;
   if (A->setsize !=B->setsize)
      Error(“…..”);
   for(i=0; i<A->arraysize; i++)
      if(A->v[i] !=B->v[i])
        { retval=0; break;}
   return retval;
}
```

- 通过集合**A**和**B**的位向量按位或来实现并集运算

```
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;
 }
```

- 通过集合**A**和**B**的位向量按位与来实现交集运算

```
Set SetIntersection(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;
 }
```

- 通过集合**A**和**B**的位向量按位与和按位异或来实现差集运算

```
Set SetDifference(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] & A->v[i]);
   return tmp;
 }
```

- 通过将集合**S**的位向量相应位置**1**来实现元素插入运算

```
void SetInsert(int x, Set S)
{
   if (x<0 || x>=S->setsize)
      Error(“…..”);
   S->v[ArrayIndex(x)] |=BitMask(x);
 }
```

- 通过清除集合**S**的位向量相应位来实现元素删除运算

```
void SetDelete(int x, Set S)
{
   if (x<0 || x>=S->setsize)
      Error(“…..”);
   S->v[ArrayIndex(x)] &=~BitMask(x);
 }
```

8.3 用有序链表实现集合

- 用链表来表示集合时，在链表中存放的是集合中元素的实际数值，而不是元素是否属于集合的标记。
- 链表中的每个结点表示集合中的一个元素。
- 因为我们讨论的是有序集，如果将集合中的所有元素按“＜”关系排序构造有序链表，给集合的某些运算会带来方便。

- **LinkedSet**的有序链表结点类型**Node**的定义

```
typedef struct node *link
struct node {
    ListItem element;
    link next;
} Node;
```

- 有序链表实现的集合**Set**的定义

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

- 创建一个空集合

```
Set SetInit( )
{
   Set S=malloc(sizeof *S);
   S->first=0;
   return S;
 }
```

- 判断集合**S**是否为空

```
int SetEmpty(Set S)
{
   return S->first==0;
 }
```

- 返回集合**S**的大小

```
int SetSize(Set S)
{
   int len;
   link current;
   current=S->first;
   len=0;
   while (current){
         len++;
         current=current->next;
       }
   return len;
 }
```

- 通过复制表示集合的链表来实现赋值运算

```
void SetAssign(Set A, Set B)
{
  link a,b,c;
  b=B->first;
  A->first=0;
  if (b){
        A->first=NewNode( );
        a=A->first;
        a->element=b->element;
        a->next=0;
        b=b->next;
        }
  while(b){
       c=NewNode( );
       c->element=b->element;
       c->next=0;
       b=b->next;
       a->next=c;
       a=c;
        }
 }
```

- 通过扫描表示集合**A**和**B**的链表来实现交集运算

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

- 通过向表示集合**S**的链表插入元素**x**来实现元素插入运算

```
void SetIntsert(ListItem x, Set S)
{
   link p,q,r;
   p = S->first;
   q = p;
   while ( p && p->element < x) {
    q=p;
    p=p->next;
    }
    if (p && p->element == x)  return;
    r = NewNode();
    r ->element =x;
    r->next = p;
    if (p == q)  S->first = r;
    else  q->next =r;
 }
```

8.4 应用举例：Eratosthenes筛法

- P186

本章小结

- 以集合为基础的**ADT**
- 实现集合的方法
  - 位向量
  - 链表