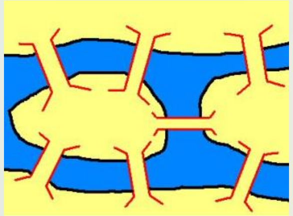
第1周小结

1 从数据结构角度求解问题的过程



问题

提取 →

数据的逻辑结构

数据运算（运算描述）

} 抽象数据类型（ADT）

ADT = 数据的逻辑结构 + 数据运算（运算描述） 逻辑层面

映射

存储结构

运算描述实现

算法

设计好存储结构使算法更优

实现层面

算法分析

好算法

分析层面

描述一个集合的抽象数据类型ASet，其中所有元素为正整数，集合的基本运算包括：

（1）由整数数组$a[0..n-1]$创建一个集合。

（2）输出一个集合的所有元素。

（3）判断一个元素是否在一个集合中。

（4）求两个集合的并集。

（5）求两个集合的差集。

（6）求两个集合的交集。

在此基础上设计集合的顺序存储结构，并实现各基本运算的算法。

解： 抽象数据类型ASet的描述如下：

```
ADT ASet
{       数据对象：D={ d_i | 0≤i≤n，n为一个正整数}
        数据关系：无。
        基本运算：
            createset( &s，a，n)：创建一个集合s;
            dispset( s)：输出集合s;
            inset(s，e)：判断e是否在集合s中
            void add(s1，s2，s3)：s3=s1∪s2;              //求集合的并集
            void sub(s1，s2，s3)：s3=s1-s2;              //求集合的差集
            void intersection(s1，s2，s3)：s3=s1∩s2;     //求集合的交集
}
```

设计集合的顺序存储结构类型如下：

```
typedef struct              //集合结构体类型
{    int data[MaxSize];     //存放集合中的元素，其中MaxSize为常量
     int length;            //存放集合中实际元素个数
} Set;                      //将集合结构体类型用一个新类型名Set表示
```

静态分配方式

采用Set类型的变量存储一个集合。对应的基本运算算法设计如下：

```
void createset(Set &s, int a[], int n)  //创建一个集合
{   int i;
    for (i=0;i<n;i++)
        s.data[i]=a[i];
    s.length=n;
}

void dispset(Set s)              //输出一个集合
{   int i;
    for (i=0;i<s.length;i++)
        printf("%d ", s.data[i]);
    printf("\n");
}
```

```
bool inset(Set s， int e)          //判断e是否在集合s中
{    int i;
     for (i=0;i<s.length;i++)
          if (s.data[i]==e)
               return true;
     return false;
}
```

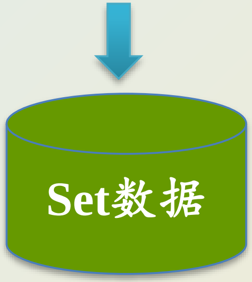
```
void add(Set s1， Set s2， Set &s3)     //求集合的并集
{     int i;
      for (i=0;i<s1.length;i++)               //将集合s1的所有元素复制到s3中
            s3.data[i]=s1.data[i];
      s3.length=s1.length;
      for (i=0;i<s2.length;i++)               //将s2中不在s1中出现的元素复制到s3中
            if (!inset(s1， s2.data[i]))
            {     s3.data[s3.length]=s2.data[i];
                  s3.length++;
            }
}
```

```
void sub(Set s1，Set s2，Set &s3)     //求集合的差集
{    int i;
     s3.length=0;
     for (i=0;i<s1.length;i++)          //将s1中不出现在s2中的元素复制到s3中
          if (!inset(s2，s1.data[i]))
          {    s3.data[s3.length]=s1.data[i];
               s3.length++;
          }
}
```

```
void intersection(Set s1，Set s2，Set &s3)      //求集合的交集
{    int i;
     s3.length=0;
     for (i=0;i<s1.length;i++)              //将s1中出现在s2中的元素复制到s3中
         if (inset(s2，s1.data[i]))
         {    s3.data[s3.length]=s1.data[i];
              s3.length++;
         }
}
```

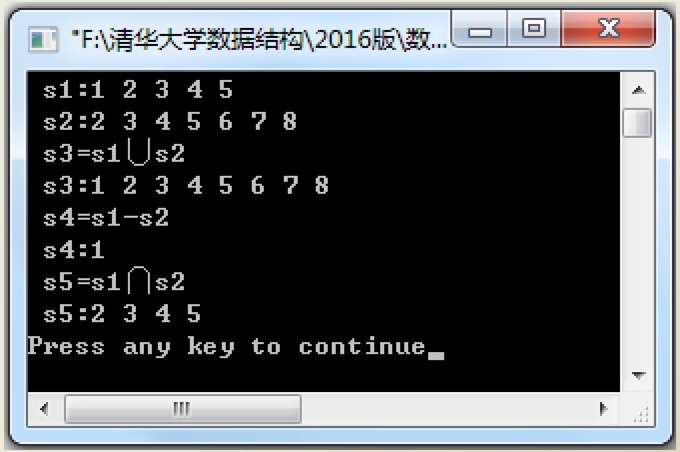
集合数据结构（已实现）

- createset(Set &s，int a[]，int n)：创建一个集合
- dispset(Set s)：输出一个集合
- inset(Set s，int e)：判断e是否在集合s中
- add(Set s1，Set s2，Set &s3)：求集合的并集
- sub(Set s1，Set s2，Set &s3)：求集合的差集
- intersection(Set s1，Set s2，Set &s3)：求集合的交集



集合数据结构的应用

```
void main()
{    Set s1， s2， s3， s4， s5;
     int a[]={1， 2， 3， 4， 5};
     int b[]={2， 3， 4， 5， 6， 7， 8};
     int n=5， m=7;
     createset(s1， a， n);
     createset(s2， b， m);
     printf(" s1:"); dispset(s1);
     printf(" s2:"); dispset(s2);
     printf(" s3=s1∪s2\n");
     add(s1， s2， s3);
     printf(" s3:"); dispset(s3);
     printf(" s4=s1-s2\n");
     sub(s1， s2， s4);
     printf(" s4:"); dispset(s4);
     printf(" s5=s1∩s2\n");
     intersection(s1， s2， s5);
     printf(" s5:"); dispset(s5);
}
```



2

算法描述—输出型参数

算法：输入⇨ 输出

```
返回值 函数名(输入参数，  输出参数)
{
      //实现代码;
}
```

采用引用类型参数

设计一个算法求整数集合s中偶数元素个数。

集合s → 算法 → 偶数元素个数m

```
void Evennumbers(Set s， int &m)
{      m=0;
       for (int i=0;i<s.length;i++)
           if (s.data[i]%2==0)  m++;
}
```

3 算法时间复杂度分析

算法类别：

- 非递归算法
- 递归算法

❶ 非递归算法

确定问题规模n

↓

找基本操作语句

↓

求基本操作的执行次数$T(n)$

↓

用复杂度表示$T(n)$

❷ 递归算法

确定问题规模n

↓

确定终止情况

↓

确定递推情况

↓

由递推式求出$T(n)$

↓

用复杂度表示$T(n)$

递推式

有如下递归算法，分析调用max(a，0，n−1)的时间复杂度。

```
int max(int a[]， int i， int j)
{     int mid=(i+j)/2， max1， max2;
      if (i<j)
      {      max1=max(a， i， mid);
             max2=max(a， mid+1， j);
             return (max1>max2)?max1:max2;
      }
      else return a[i];
}
```

```
int max(int a[], int i, int j)
{    int mid=(i+j)/2, max1, max2;
     if (i<j)
     {      max1=max(a, i, mid);
            max2=max(a, mid+1, j);
            return (max1>max2)?max1:max2;
     }
     else return a[i];
}
```

设调用max(a, 0, n−1)的执行时间为$T(n)$

递归算法max(a, i, j)的执行时间为$T_1(n)$（$n=j-i+1$）

有$T(n)=T_1(n)$

递推式

$T(n)=O(1)$	当$n=1$（$i=j$的情况）
$T(n)=2T(n/2)+1$	当$n>1$（$i<j$的情况）

$$T(n)=O(1) \quad \text{当} n=1$$
$$T(n)=2T(n/2)+1 \quad \text{当} n>1$$

$$T(n) = 2T(n/2) + 1$$
$$= 2[2T(n/2^2) + 1] + 1 = 2^2T(n/2^2) + 2 + 1$$
$$= \cdots$$
$$= 2^kT(n/2^k) + 2^{k-1} + \cdots + 2 + 1 \ (k=\log_2 n)$$
$$= 2^k + 2^{k-1} + \cdots + 2 + 1$$
$$= 2*2^k - 1$$
$$= O(n)$$

调用max(a，0，n−1)的时间复杂度为O(n)