2.5 有序表

1、什么是有序表

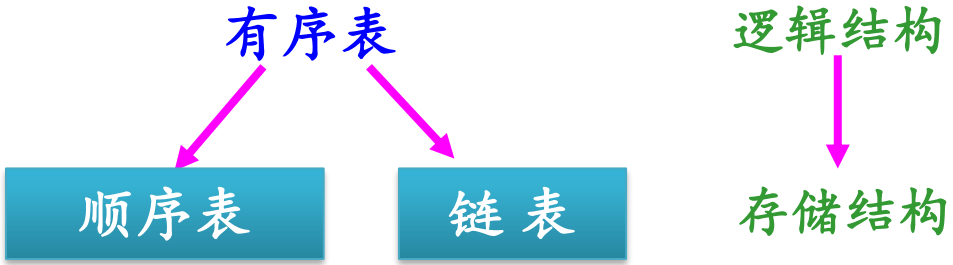
所谓**有序表**，是指这样的线性表，其中所有元素以递增或递减方式有序排列。

有序表 ⊂ 线性表

为了简单，假设有序表元素是以递增方式排列。

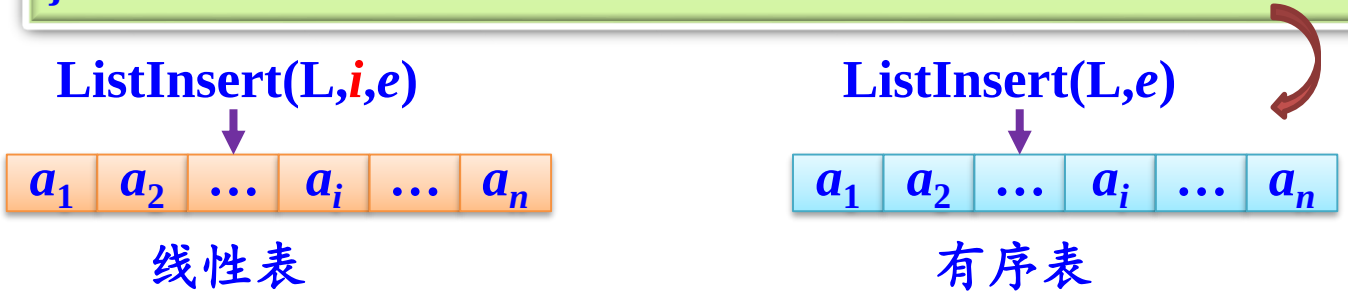
从中看到，有序表和线性表中元素之间的逻辑关系相同，其区别是运算实现的不同。

既然有序表中元素逻辑关系与线性表的相同，有序表可以采用与线性表相同的存储结构。



若以顺序表存储有序表，会发现基本运算算法中只有ListInsert()算法与前面的顺序表对应的运算有所差异，其余都是相同的。有序顺序表的ListInsert()算法如下：

```
void ListInsert(SqList *&L,ElemType e)
{    int i=0,  j;
     while (i<L->length && L->data[i]<e)
            i++;                                  //查找值为e的元素
     for (j=ListLength(L);j>i;j--)                //将data[i..n]后移一个位置
            L->data[j]=L->data[j-1];
     L->data[i]=e;
     L->length++;                                 //有序顺序表长度增1
}
```



若以单链表存储有序表，同样发现基本运算算法中只有ListInsert()算法与前面的单链表对应的运算有所差异，其余都是相同的。有序单链表的ListInsert()的算法如下：

```
void ListInsert(LinkList *&L,ElemType e)
{    LinkList *pre=L,*p;

     while (pre->next!=NULL && pre->next->data<e)
            pre=pre->next;      //查找插入节点的前驱节点*pre

     p=(LinkList *)malloc(sizeof(LinkList));
     p->data=e;                 //创建存放e的数据节点*p
     p->next=pre->next;         //在*pre节点之后插入*p节点
     pre->next=p;

}
```

查找插入的位置*pre

在*pre之后插入*p

思考题：

有序表和线性表有什么异同？

有序表和顺序表有什么不同？

2、有序表的归并算法

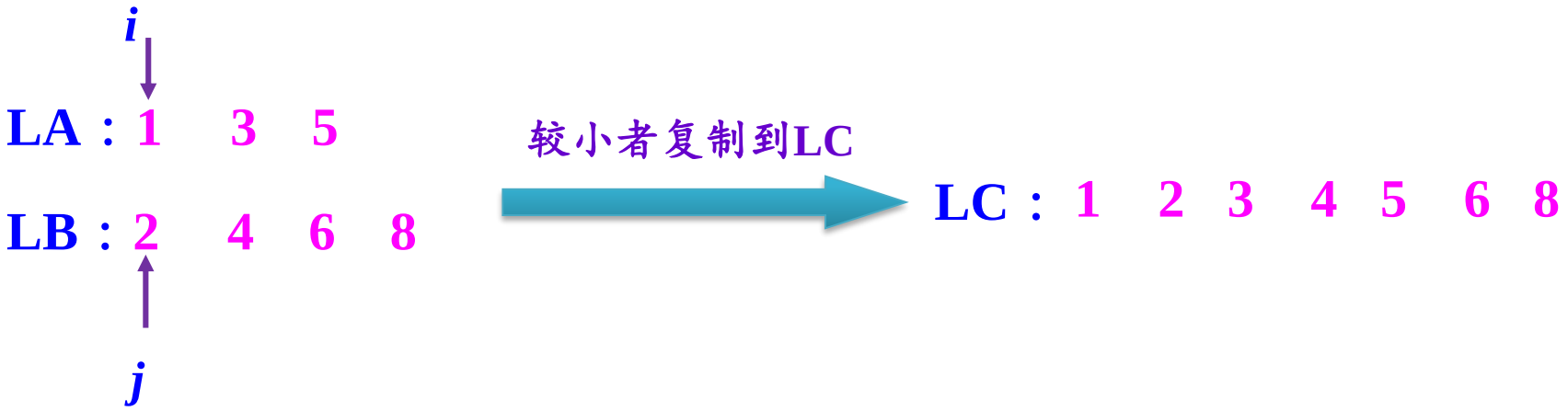
【例2-11】 假设有两个有序表LA和LB。设计一个算法，将它们合并成一个有序表LC。

二路归并示意图

顺序表二路归并示例的演示

例如，LA=（1，3，5），LB=（2，4，6，8），

其二路归并过程如下：



LA、LB中每个元素恰好遍历一次，时间复杂度为O(*m*+*n*)。

采用顺序表存放有序表时，二路归并算法如下：

```
void UnionList(SqList *LA,SqList *LB,SqList *&LC)
{    int i=0, j=0, k=0;          //i、j分别为LA、LB的下标,k为LC中元素个数
     LC=(SqList *)malloc(sizeof(SqList));        //建立有序顺序表LC
     while (i<LA->length && j<LB->length)
     {       if (LA->data[i]<LB->data[j])
             {       LC->data[k]=LA->data[i];
                     i++; k++;
             }
             else                    //LA->data[i]>LB->data[j]
             {      LC->data[k]=LB->data[j];
                    j++; k++;
             }
      }
```

两个有序表均没有遍历完

```
    while (i<LA->length)                //LA尚未扫描完,将其余元素插入LC中
    {       LC->data[k]=LA->data[i];
            i++;k++;
    }
    while (j<LB->length)                //LB尚未扫描完,将其余元素插入LC中
    {       LC->data[k]=LB->data[j];
            j++;k++;
    }
    LC->length=k;
}
```

本算法的时间复杂度为O(*m*+*n*)，空间复杂度为O(*m*+*n*)。

采用单链表存放有序表时，二路归并算法如下：

```
void UnionList1(LinkList *LA,LinkList *LB,LinkList *&LC)
{    LinkList *pa=LA->next, *pb=LB->next, *r, *s;
     LC=(LinkList *)malloc(sizeof(LinkList));   //创建LC的头节点
     r=LC;                                      //r始终指向LC的尾节点
     while (pa!=NULL && pb!=NULL)
     {       if (pa->data<pb->data)
             {    s=(LinkList *)malloc(sizeof(LinkList));    //复制节点
                  s->data=pa->data;
                  r->next=s;  r=s;                          //采用尾插法将*s插入到LC中
                  pa=pa->next;
             }
             else
             {    s=(LinkList *)malloc(sizeof(LinkList));    //复制节点
                  s->data=pb->data;
                  r->next=s;  r=s;                          //采用尾插法将*s插入到LC中
                  pb=pb->next;
             }
     }
```

```
    while (pa!=NULL)
    {    s=(LinkList *)malloc(sizeof(LinkList));   //复制节点
         s->data=pa->data;
         r->next=s;r=s;          //采用尾插法将*s插入到LC中
         pa=pa->next;
    }
    while (pb!=NULL)
    {    s=(LinkList *)malloc(sizeof(LinkList));   //复制节点
         s->data=pb->data;
         r->next=s;r=s;          //采用尾插法将*s插入到LC中
         pb=pb->next;
    }
    r->next=NULL                 //尾节点的next域置空
}
```

若LA没有扫描完，将余下节点复制到LC中

若LB没有扫描完，将余下节点复制到LC中

本算法的时间复杂度为O(*m*+*n*)，空间复杂度为O(*m*+*n*)。

思考题：

假设两个有序表LA和LB采用单链表存储，设计一个算法，将它们合并成一个有序表单链表LC。要求算法的空间复杂度为O(1)。

【例2-12】一个长度为L（L≥1）的升序序列S，处在第⌈L/2⌉个位置的数称为S的中位数。

例如：若序列S_1=(11,13,15,17,19)，则S1的中位数是15。

两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若S_2=(2,4,6,8,20)，则S_1和S_2的中位数是11。

现有两个等长的升序序列A和B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列A和B的中位数。要求：

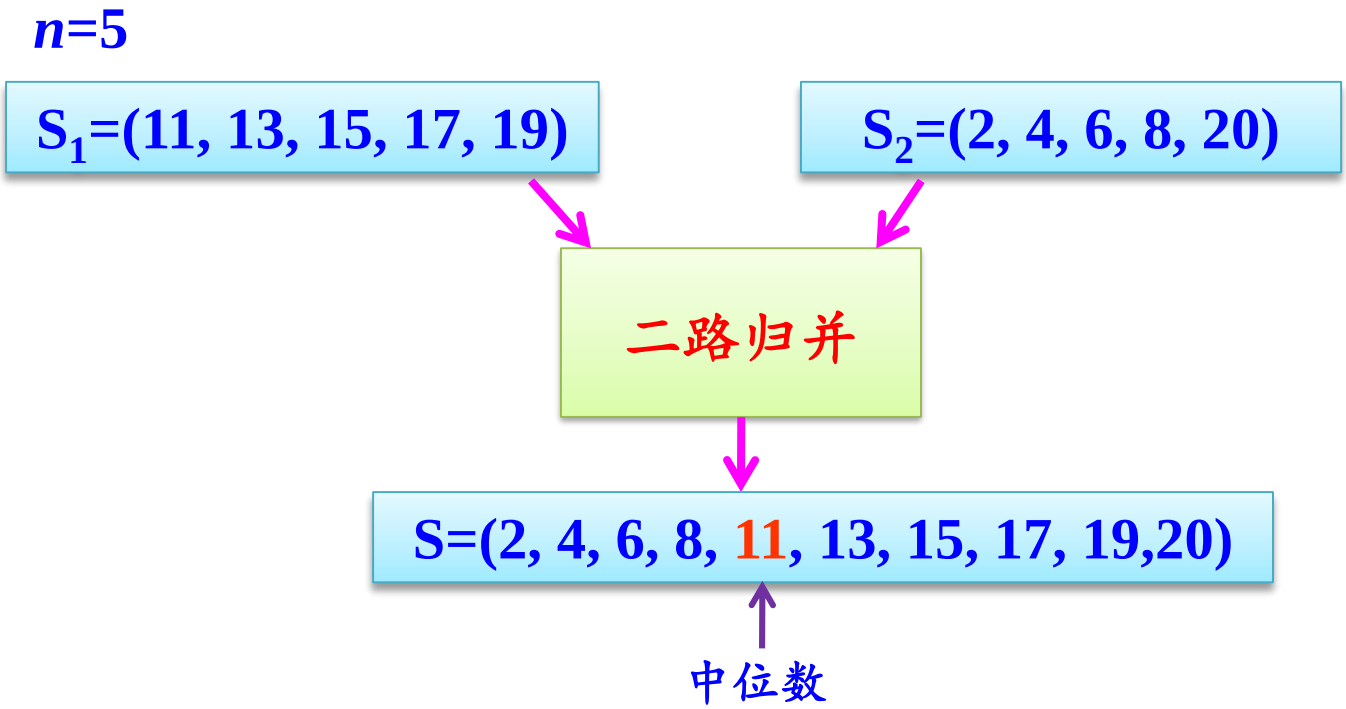
（1）给出算法的基本设计思想。

（2）根据设计思想，采用C、C++或Java语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

注：本题为2011年全国考研题。

算法设计思路



实际上，不需要求出S的全部元素，用*k*记录当前归并的元素个数，当*k*=*n*时，归并的那个元素就是中位数。

$n=5$

i ↓

$S_1=(11, 13, 15, 17, 19)$

$S_2=(2, 4, 6, 8, 20)$

↑ j

$k=5$

$k=n$，结果为i、j所指较小的元素

中位数为$S_1[i]=11$

对应的算法如下：

```
int M_Search(int A[], int B[], int n)
{     int i,  j,  k;
      i = j = k = 0;
      while (i<n && j<n)
       {      k++;
              if (A[i]<B[j])
              {     if (k==n)  return A[i];
                     i++;
              }
              else        //A[i]>=B[j]
              {     if (k==n)  return B[j];
                    j++;
             }
      }
}
```

算法的时间复杂度为O(*n*)，空间复杂度为O(1)。

——本章完——