第2周小结

知识点：

- 线性表概念
- 顺序表及算法设计
- 单链表及算法设计

线性表两类存储结构的比较

I. 顺序表

II. 链表

❶ 顺序表

优点

- 存储密度大：无须为表示线性表中元素之间的逻辑关系而增加额外的存储空间。
- 具有随机存取特性。

缺点

- 插入和删除操作需要移动大量元素。
- 初始空间大小分配难以掌握。

② 链表

优点

- 由于采用节点的动态分配方式，具有良好的适应性。
- 插入和删除操作只需修改相关指针域，不需要移动元素。

缺点

- 存储密度小：为表示线性表中元素之间的逻辑关系而需要增加额外的存储空间（指针域）。
- 不具有随机存取特性。

2 线性表的算法设计

一般算法如何设计？

- 数据的存储结构—顺序表：链表？
- 算法的处理过程—用C/C++语言描述。

（1）顺序表算法设计

注意：

- 顺序表—用数组表示 ⇨ 借鉴数组处理方法（存、取元素）
- 顺序表—不同于数组 ⇨ 顺序表是线性表的一种存储结构

线性表L：（1，2，3）

数组：int a[]={1，2，3}；

而数组：int b[]={空，1，2，空，3}；不对应L

❶ 基于顺序表基本操作的算法设计

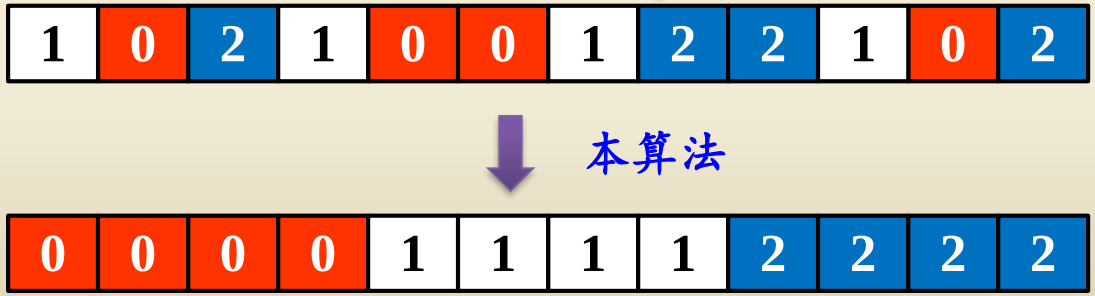
- 查找元素
- 插入元素
- 删除元素

❷ 基于特殊方法的顺序表算法设计

- 将整数顺序表L以第一个元素为分界线（基准）进行划分
- 在顺序表L中删除所有值为x的元素
- ……

荷兰国旗问题：设有一个条块序列，每个条块为红（0）、白（1）、兰（2）三种颜色中的一种。假设该序列采用顺序表存储，设计一个时间复杂度为O(*n*)的算法，使得这些条块按红、白、兰的顺序排好，即排成荷兰国旗图案。

例如：1 0 2 1 0 0 1 2 2 1 0 2

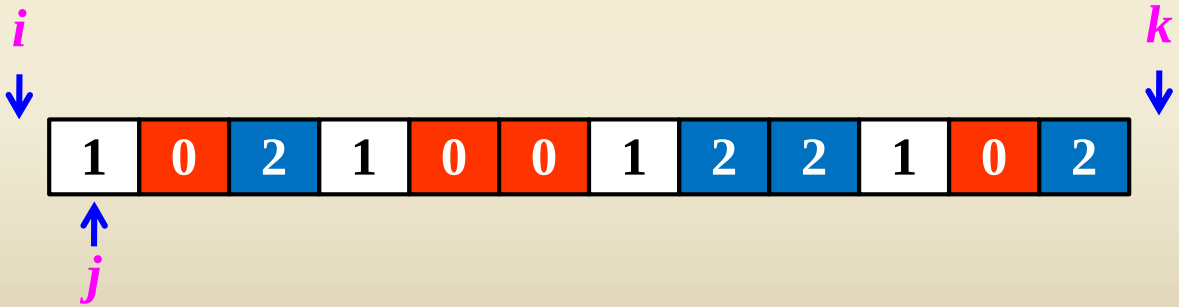


解：

- 用0～i表示0元素区间。
- k～$n-1$表示2元素区间。
- 中间部分为1元素区间。
- 用j从头开始扫描顺序表L中部的所有元素。

每一次循环：

- *j*指向元素1：说明它属于中部，保持不动，*j*++。
- *j*指向元素0：说明它属于前部，*i*增1（扩大0元素区间），将*i*、*j*位置的元素交换，*j*++。
- *j*指向元素2：说明它属于后部，*k*减1（扩大2元素区间），将*j*、*k*位置的元素交换，此时*j*位置的元素可能还要交换到前部，所以*j*不前进。



*j*指向0，交换到前面 …

算法如下：

```
void move1(SqList *&L)
{    int i=-1，j=0，k=L->length;
     while (j<k)
     {       if (L->data[j]==0)
             {     i++;
                   swap(L->data[i]，L->data[j]);
                   j++;
             }
             else if (L->data[j]==2)
             {     k--;
                   swap(L->data[k]，L->data[j]);
             }
             else  j++;   //L->data[j[==1的情况
     }
}
```

（2）单链表算法设计

❶ 基于单链表基本操作的算法设计

- 查找节点
- 插入节点
- 删除节点

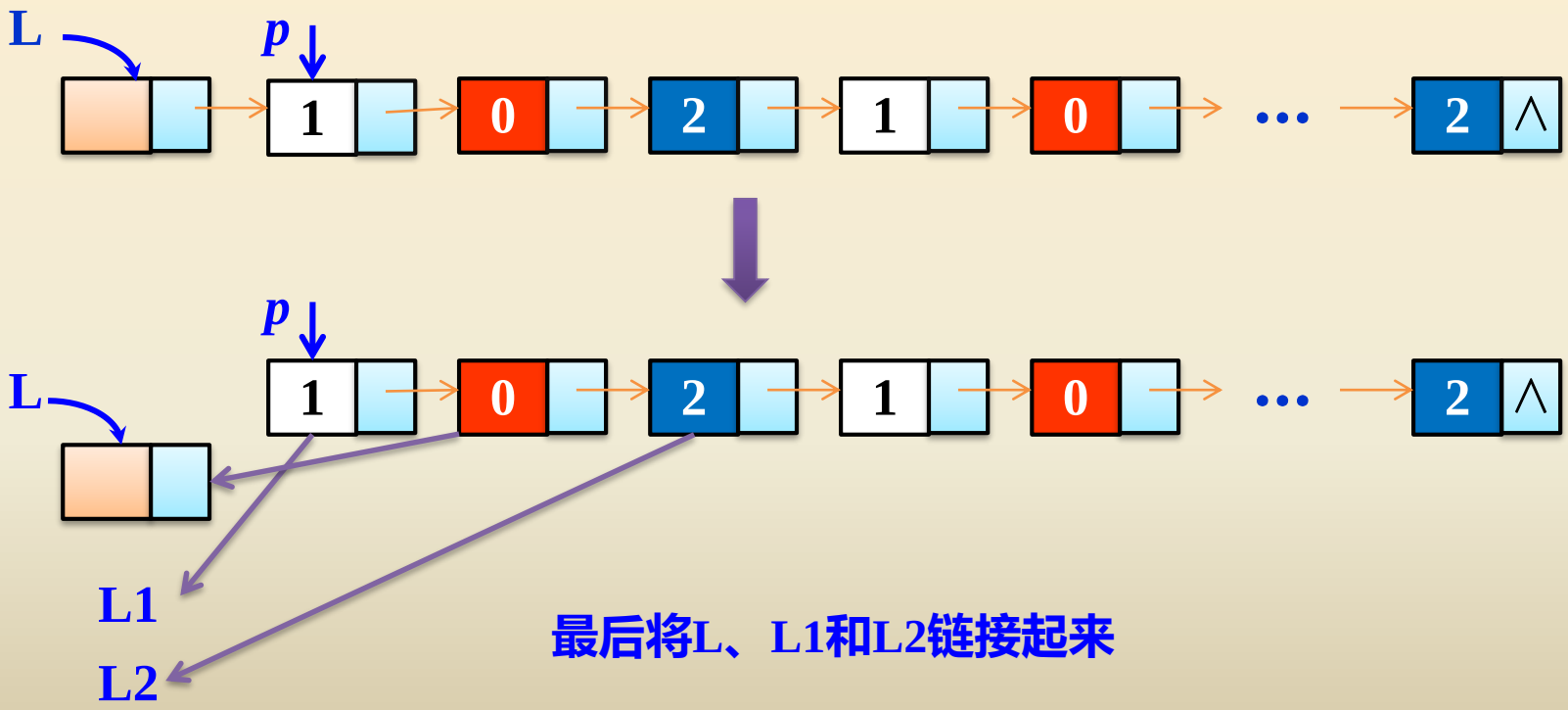
} 需要查找前驱节点

❷ 基于两个建表方法的单链表算法设计

- 头插法：相对次序相反
- 尾插法：相对次序相同

荷兰国旗问题：设有一个仅由红（0）、白（1）、兰（2）这三种颜色的条块组成的条块序列。假设该序列采用单链表存储，设计一个时间复杂度为O(*n*)的算法，使得这些条块按红、白、兰的顺序排好，即排成荷兰国旗图案。

解： 用p指针扫描节点，根据p->data值将该节点插入到3个单链表L、L1和L2（L1和L2不带头节点的）中。最后将它们链接起来。



算法如下：

```
void move2(LinkList  *&L)
{    LinkList *L1，L2，*r，*r1，*r2，*p;
     L1=NULL;
     L2=NULL;
     p=L->next;
     r=L;
```

做准备工作

```
while (p!=NULL)
{
    if (p->data==0)
    {   r->next=p; r=p;  }

    else if (p->data==1)
    {
        if (L1==NULL)
        {   L1=p; r1=p;   }
        else
        {   r1->next=p; r1=p;  }
    }
    else            //p->data==2
    {
        if (L2==NULL)
        {   L2=p; r2=p;   }
        else
        {   r2->next=p; r2=p;  }
    }
    p=p->next;
}
```

建立L带头节点的单链表

建立L1不带头节点的单链表

建立L2不带头节点的单链表

```
    r->next=r1->next=r2->next=NULL;
    r->next=L1;     //L的尾节点和L1的首节点链接起来
    r1->next=L2;   //L1的尾节点和L2的首节点链接起来
}
```

结尾工作

所以，两个建表算法是许多算法设计的基础！