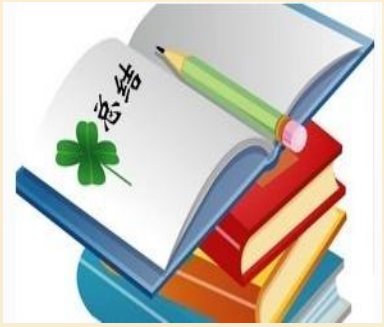


第2章 小结

一 顺序表和单链表

1、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。

❷ 链表

优点

- 由于采用结点的动态分配方式，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、选择适合的数据结构

示例 假设一个学生年级有若干个班，每个班有唯一的班号(如 1班、2班等)，一个班有若干学生(人数可能从 10到几百)，每个学生信息包括学号和姓名(所有学生的班号和学号均唯一)，一个班的学生 记录的排列是无序的。其中最 频繁的操作如下：

① 添加一个某班某学号和姓名的学生 记录。

② 删除某班某学号的学生 记录。

③ 查找某班某学号的学生 记录。

回答以下 问题：

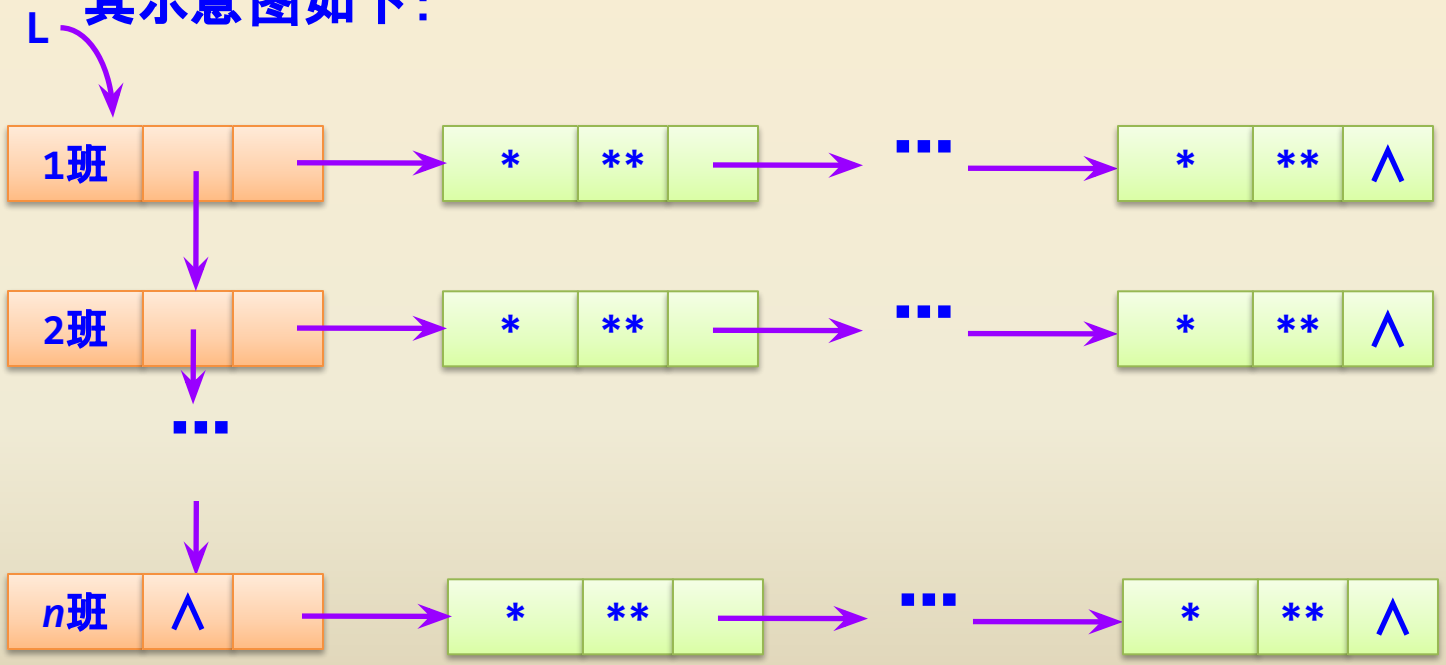
(1)设计一个你认为最合适的存 储结构用于存 储该年级所有学生信息，给出相关数据 类型的声明，并画出相 应的示意图，要求所有学生信息用一个变量标识。

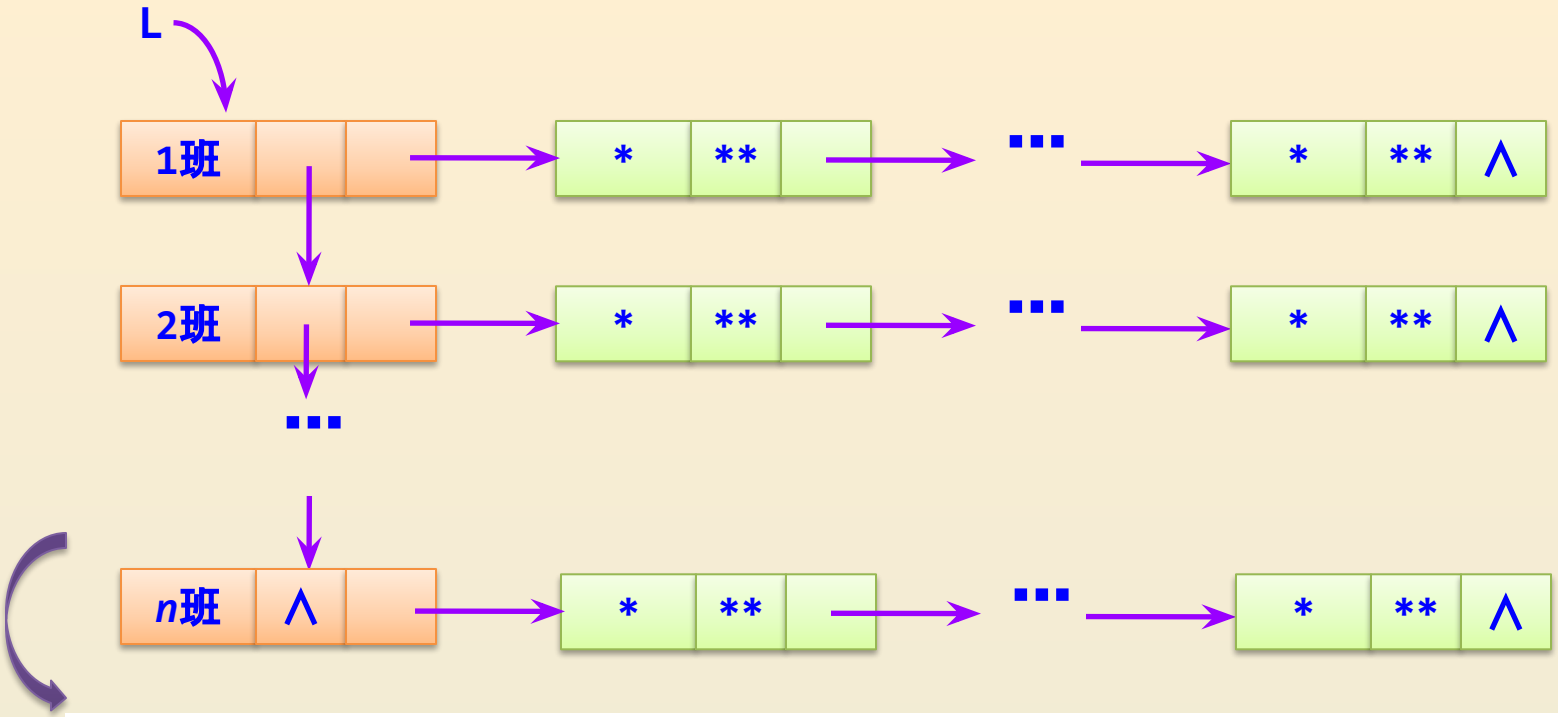
(2)假设学生班数为n，最多班的人数 为m，用文字描述或者代 码描述操作①的过程，并说明其时间 复杂度。

(1)设计一个你认为最合适的存储结构用于存储该年级所有学生信息。

(1)每个学生信息用一个结点存储，一个班的学生信息构成一个带头结点的单链表，头结点包含班号，所有班的头结点构成一个单链表，用其头指针标识整个存储结构。

其示意图如下：

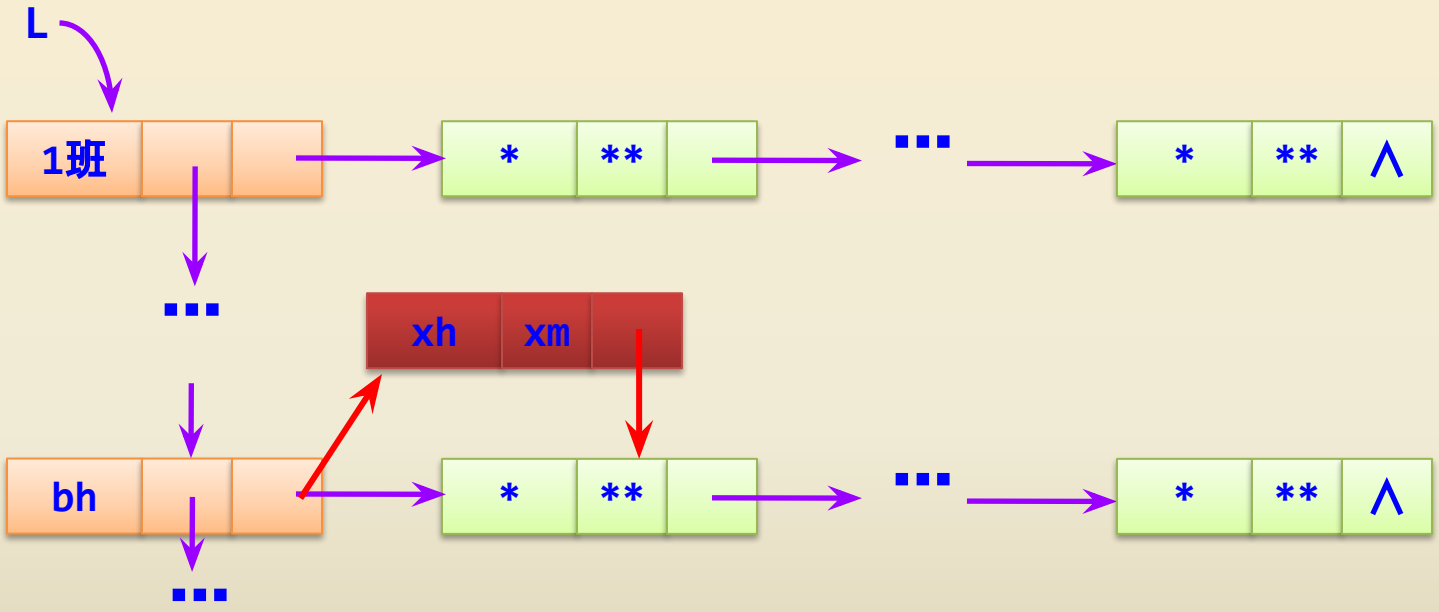





```
typedef struct node
{   string no;          //学号或者班号
    union
    {   string name;        //姓名
        struct node *bp;    //班号指针
    } val;
    struct node link;       //后继结点指针
} NodeType;
```

（2）假设学生班数为n，最多班的人数为m，用文字描述或者代码描述操作①的过程，并说明其时间复杂度。

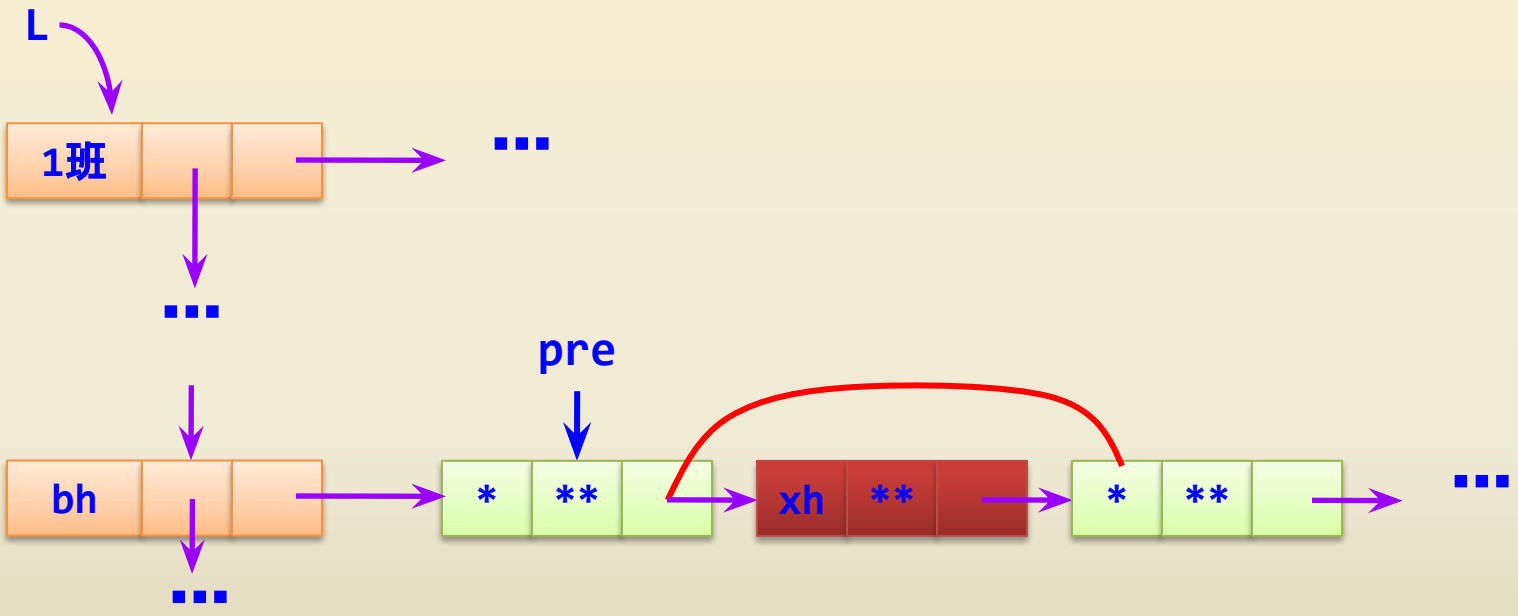
（2）创建一个插入学生信息的结点(bh,xh,xm)，通过L找到指定班bh的头结点，在其后插入该结点(头插法)。



时间复杂度为$O(n)$。

（3）假设学生班数为n，最多班的人数为m，用文字描述或者代码描述操作②的过程，并说明其时间复杂度。

（3）通过L找到指定班bh的头结点，找到该学号为xh结点的前驱结点pre，通过pre删除其后结点。



时间复杂度为O(n+m)。

线性表的算法设计

一般算法如何设计？

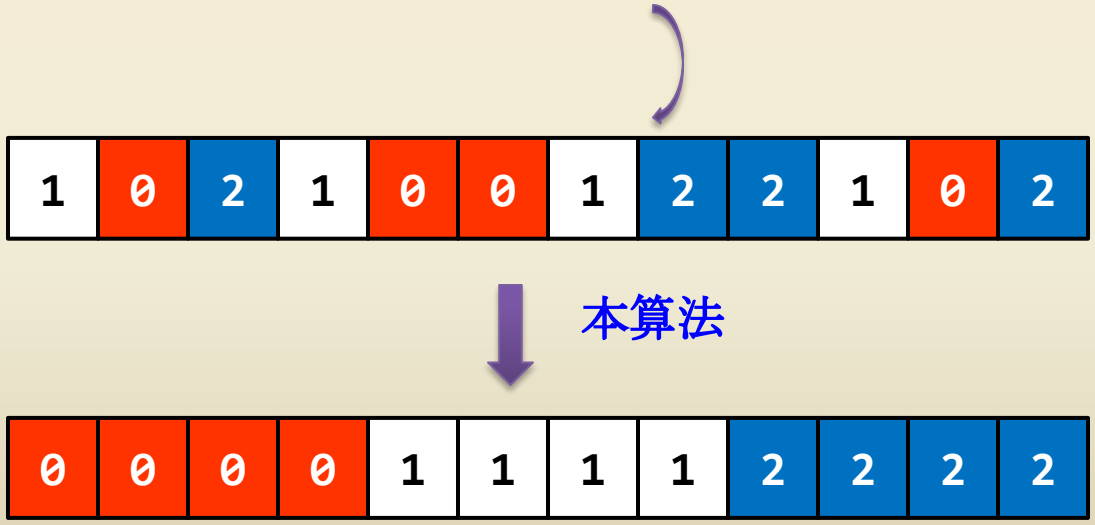
- 数据的存储结构 — 顺序表 or 链表？
- 算法的处理过程 — 用C/C++语言描述。

3、顺序表和单链表算法设计

示例

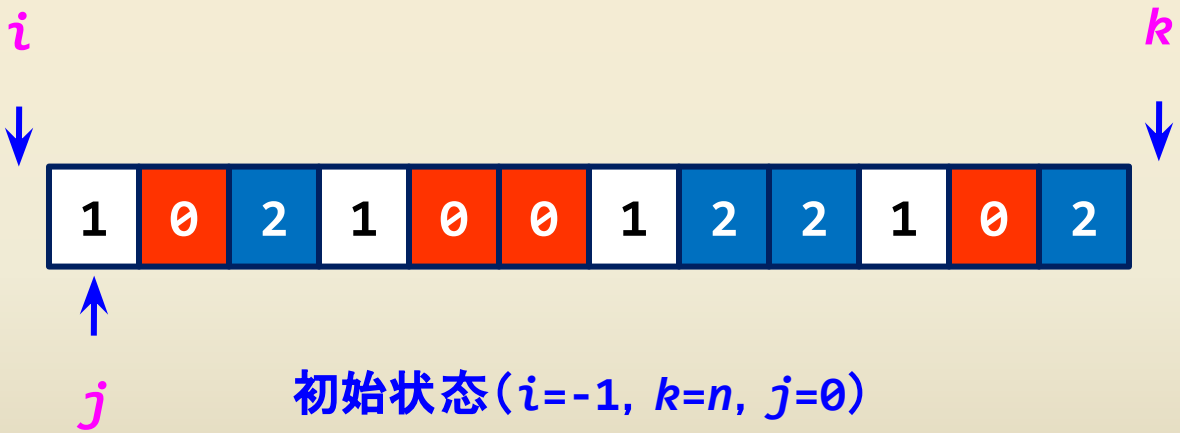
荷兰国旗问题：设有一个条块序列，每个条块为红(0)、白(1)、兰(2)三种颜色中的一种。假设该序列采用顺序表存储，设计一个时间复杂度为O(n)的算法，使得这些条块按红、白、兰的顺序排好，即排成荷兰国旗图案。

例如：1 0 2 1 0 0 1 2 2 1 0 2



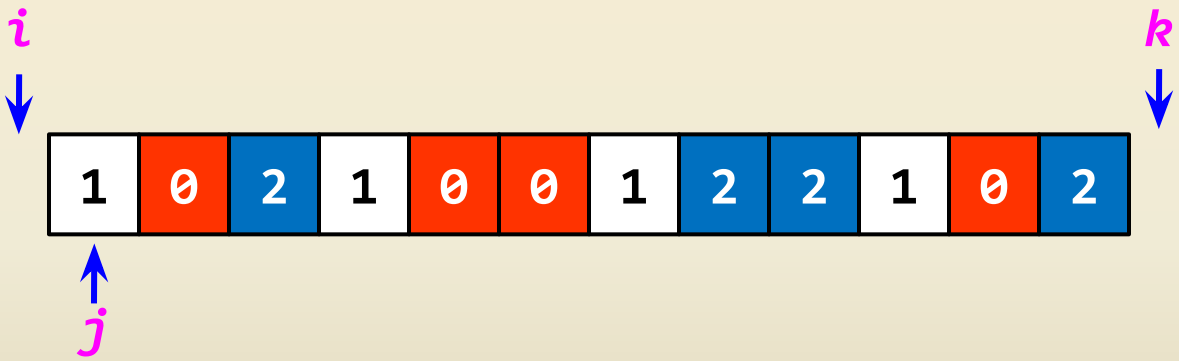
解:区间划分法

- 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 …

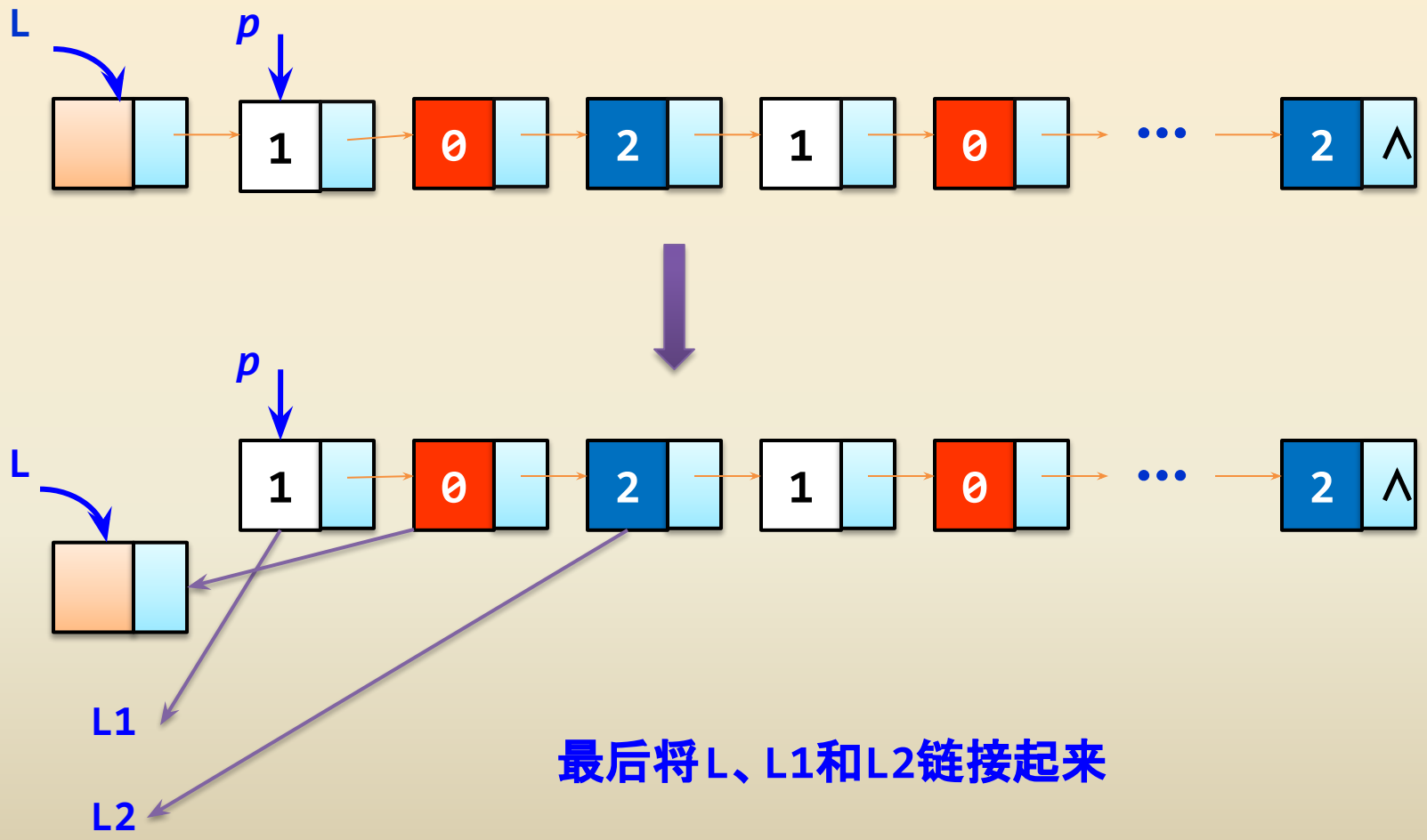
算法如下：

```
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
   }
}
```

示例

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*指针扫描结点，根据p->data值将该结点插入到3个单链表L、L1和L2（L1和L2不带头结点的）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带头结点的单链表

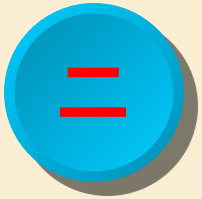
建立L1不带头结点的单链表

建立L2不带头结点的单链表

```
    r->next=r1->next=r2->next=NULL;
    r->next=L1;      //L的尾结点和L1的首结点链接起来
    r1->next=L2;         //L1的尾结点和L2的首结点链接起来
}
```

结尾工作

所以，两个建表算法是 许多算法设计的基础！



其他链表

- 双链表
- 循环链表
- 有序表

1、双链表

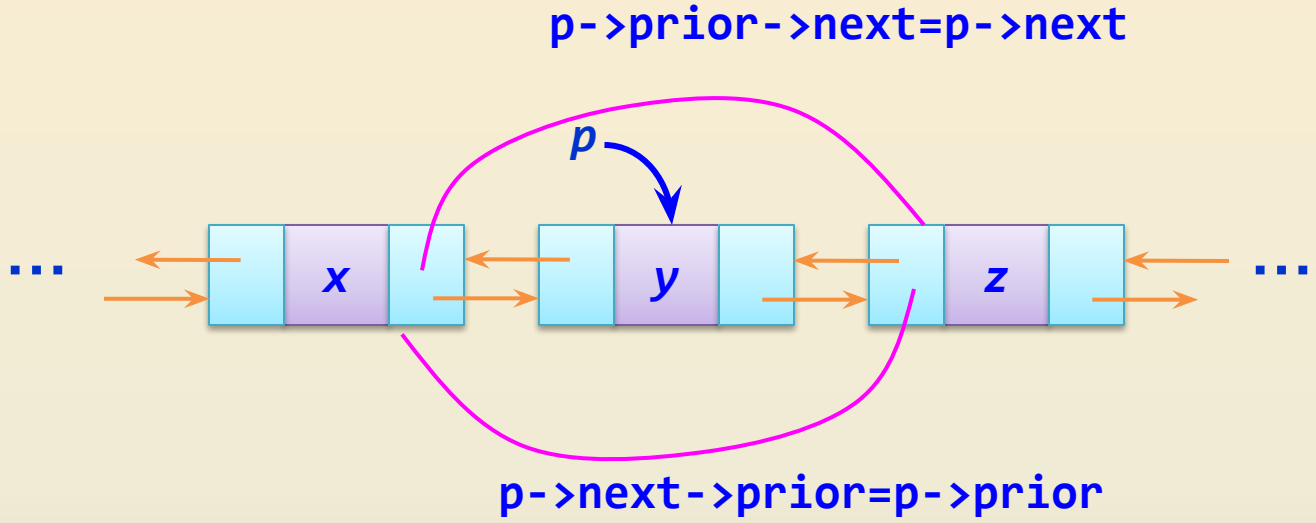
❶ 每个结点有指向前、后相 邻结点的指针域。

通常，存储密度低于单链表

❷ 特点:方便查找一个结点的前、后相邻结点。

- 已知某个结点的地址，删除它的时间为O(1)。

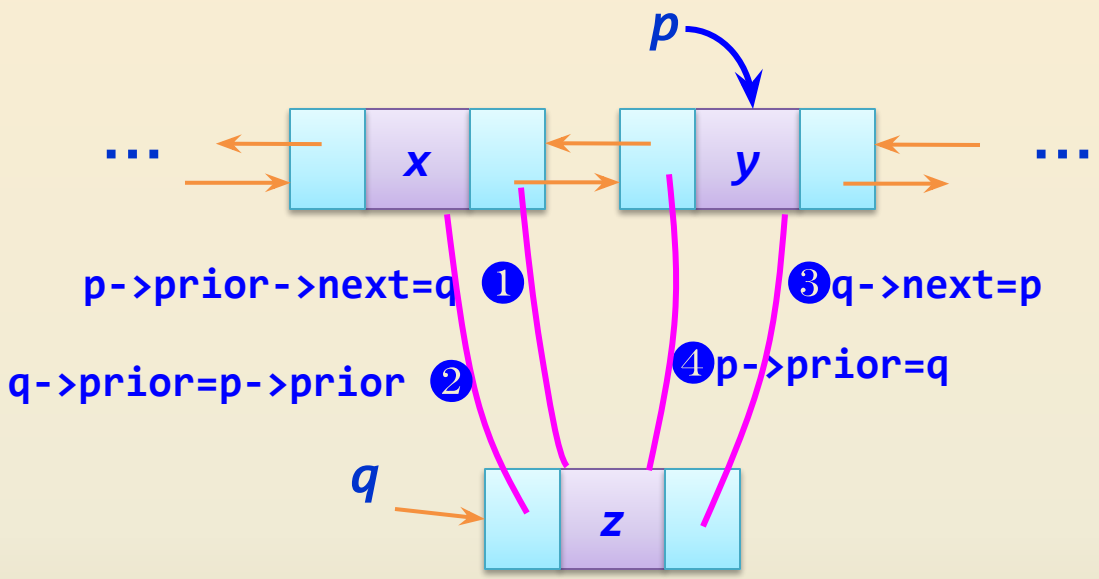
删除过程:



修改p结点前驱结点的next指针和p结点后继结点的prior指针。

- 在某个结点的前、后插入一个 结点的时间为O(1)。

前面插入过程：



通常修改p->prior在最后进行！

2、循环链表

❶ 循环单链表:构成一个环。

可以循环查找

❷ 循环双链表:构成两个 环。

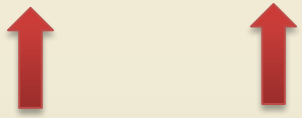
- 可以循环查找
- 可以通过头结点快速找到尾 结点

删除尾结点、在尾结点前后插入一个结点的时间均为O(1)。

3、有序表

❶ 从逻辑结构看，有序表是线性表的一个子集。

可以采用顺序表或者链表存储



有序顺序表　　有序链表

❷ 利用有序表的有序特性可以提高相关算法的效率

示例 假设一个有序表采用顺序表存储。设计一个高效算法 删除重复的元素。

例如：L=(1, 1, 1, 2, 2, 3)

本算法

L=(1, 2, 3)

解:利用前面介绍过的删除所有值为x的元素的算法思路:

```
void deldupnode1(SqList *&L)
{   int k=1,i;      //k记录保留的元素个数
    for  (i=1;i<L->length;i++)
      if (L->data[i]!=L->data[i-1])
      {
      L->data[k]=L->data[i];
      k++;       //保留的元素增1
      }
    L->length=k;    //顺序表L的长度等于k
}
```

重建顺序表L

❸ 利用二路归并思路可以提高相关算法的效率

示例

假设两个递增有序表采用单链表ha和hb存储(假设同一个单链表中不存在重复的元素)。设计一个高效算法求它们的公共元素，将结果存放在单链表hc中。

例如：ha=(1，2，3)，hb=(2，3，4)

↓ 本算法

hc=(2，3)

算法如下:

```
void InterSect(LinkList *ha, LinkList *hb, LinkList *&hc)
{  LinkList *pa=ha->next, *pb=hb->next, *s, *r;
   hc=(LinkNode *)malloc(sizeof(LinkNode));
   r=hc;                  //r指向尾结点
   while (pa!=NULL && pb!=NULL)
   {
      if (pa->data<pb->data) pa=pa->next;
      if (pa->data>pb->data) pb=pb->next;
      if (pa->data==pb->data)          //相同元素
      {  s=(LinkNode *)malloc(sizeof(LinkNode));
                  //复制结点
         s->data=pa->data;
         r->next=s;  r=s;
         pa=pa->next; pb=pb->next;
      }
   }
   r->next=NULL;
```

二路归并+尾插法建表