第二章 线性表

2.3 线性表的链式表示和实现

- 2.3.1 线性表的链式存储—链表
- 2.3.2 单链表基本运算的实现
- 2.3.5 静态链表
- 2.3.3 循环链表
- 2.3.4 双链表

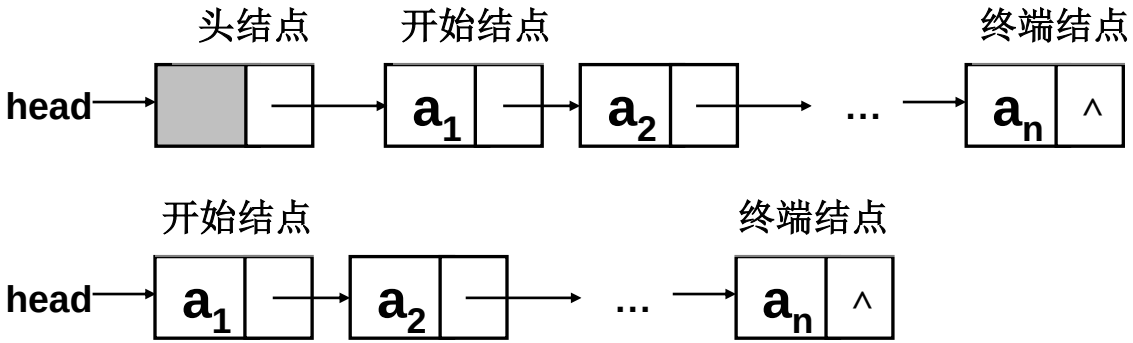
回顾

单链表的类型定义如下：

```
typedef struct LNode {
    ElemType      data;     //数据域
    struct Lnode  *next;    //指针域
}LNode, *LinkList;
```

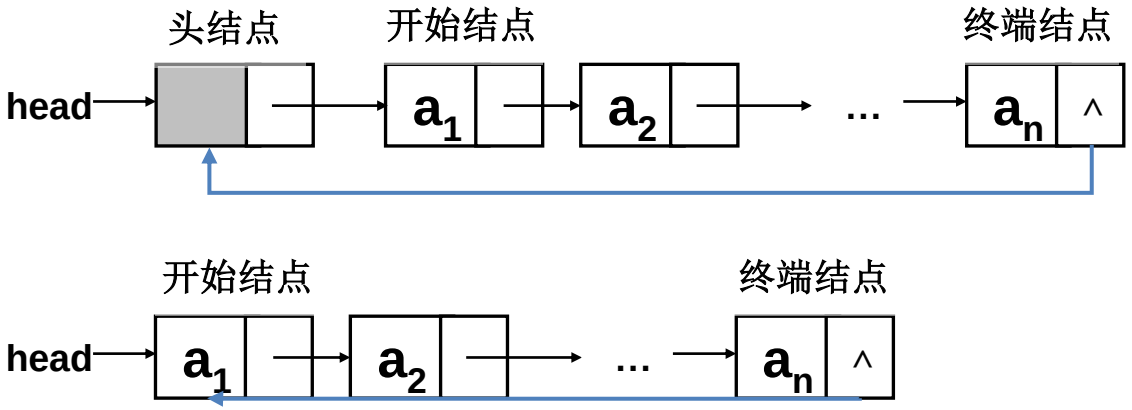
静态单链表的类型定义如下：

```
#define MAXSIZE   1000
typedef   struct  {
        ElemType   data ;    //数据域
        int            cur ;     //指示域
 }component , SLinkList[MAXSIZE] ;
```



回顾

循环链表



(1) 尾结点指针域指向头结点

(2) 结点结构与线性链表相同

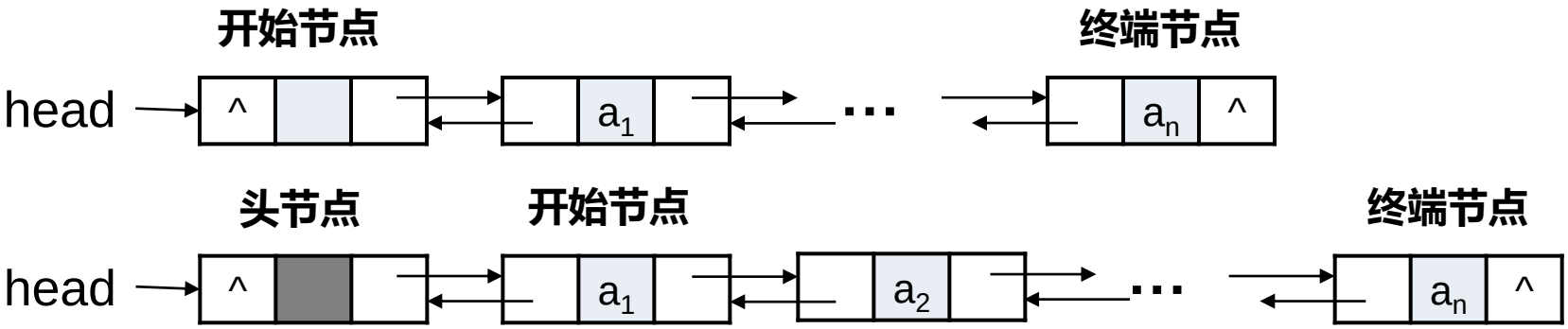
(3) 有关操作与线性链表基本相同，仅有判断链表尾方法不同

(4) 知道任何一个指向某一结点的指针就可以访问链表中的所有结点

回顾

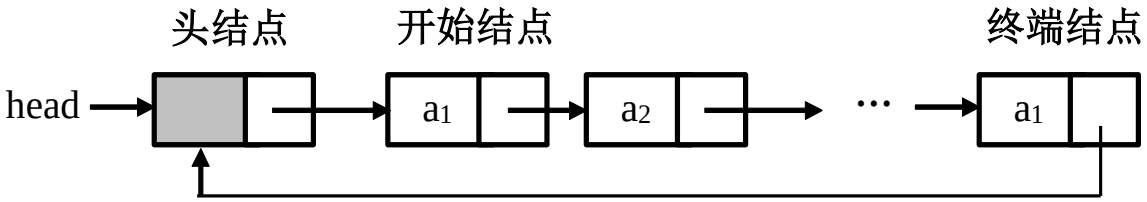
双向链表

```
typedef struct DNode   {  /*定义双链表结点类型*/
    ElemType data;
    struct DNode *prior;    /*指向前驱结点*/
    struct DNode *next;     /*指向后继结点*/
} DLinkList;
```

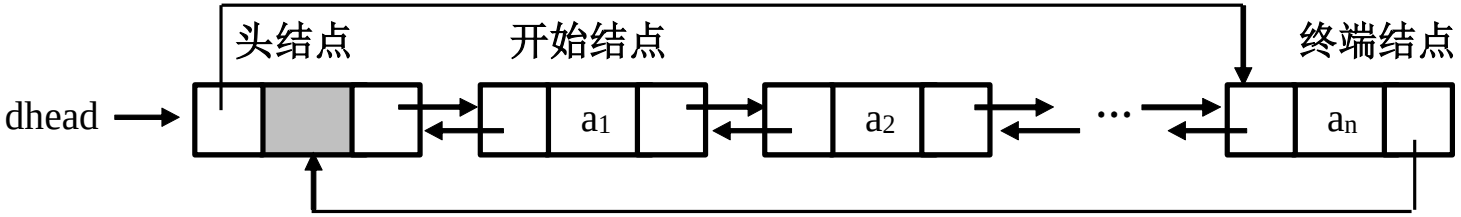


2.3.4 双链表

口 双向循环链表



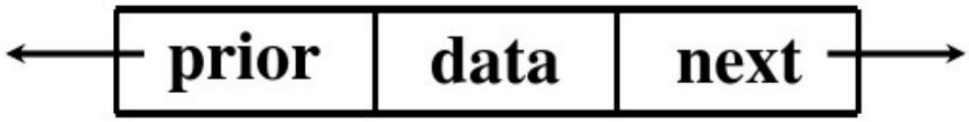
(a)循环单链表



(b)循环双链表

2.3.4 双链表

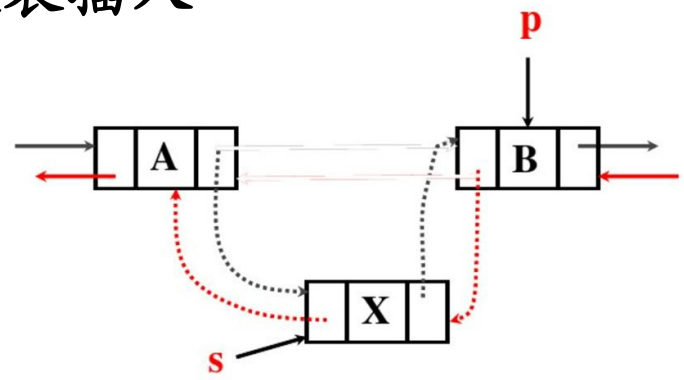
□ 双向循环链表




```
typedef   struct  DuLNode{
      ElemType              data ;
      struct  DuLNode       * prior ;
      struct  DuLNode       * next ;
} DuLNode ,  * DuLinkList ;
```

2.3.4 双链表

（1）双向链表插入



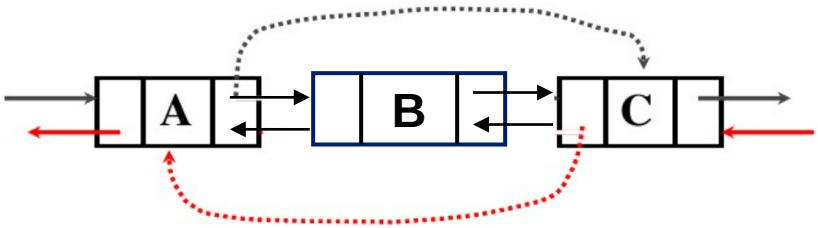
1. 找到要在之前插入的结点，p记录。
2. s->prior = p->prior ;
3. p->prior->next = s ;
4. s->next = p ;
5. p->prior = s ;

2.3.4 双链表

```
Status ListInsert_DuL(DuLinkList &L,int i,ElemType e){
    if(!(p=GetElemP_Du(L,i)))
        return ERROR;
    if(!(s=(DuLinkList)malloc(sizeof(DuNode))))
        return ERROR;
     s->data = e;
     s->prior=p->prior; p->prior->next-s;
     s->next=p; p->prior=s;
     return OK;
}
```

2.3.4 双链表

（2）双向链表删除



1. 找到要删除的结点，p记录。
2. p->prior->next = p->next ;
3. p->next->prior = p->prior ;
4. free(p) ;

2.3.4 双链表

```
Status ListDelete_ DuL(DuLinkList &L, int i, ElemType &e) {
    if (! (p = GetElemP_DuL(L, i)))
          return ERROR;
     e = p->data;
     p->prior->next = p->next;
     p->next->prior = p->prior1
     free(p);
     return OK;
}
```

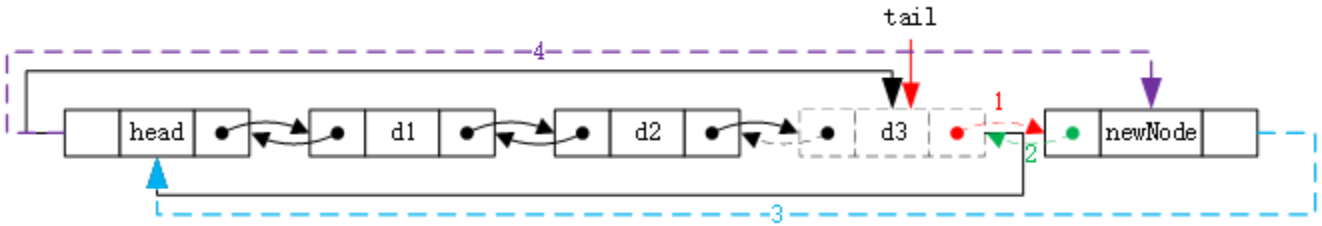
2.3.4 双链表

双向循环链表删除尾节点

```
void ListPopBack(DuLinkList L)
{
    assert(L);
    assert(L->next != L);
    DuLinkList tail = L->prior;
    DuLinkList tailPrev = tail->prior;
    tailPrev->next = L;
    L->prior = tailPrev;
    free(tail);
    tail = NULL;
}
```

2.3.4 双链表

（4）双向链表建立




```
void CreateList_DuL(DuLinkList &L){
  DuLinkList p;   int x;
  L=p=(DuLinkList)malloc(sizeof(DuLNode));
  L->next=L;   L->prior =L;
  while (scanf("%d",&x),x!=0){
    p->next=(DuLinkList)malloc(sizeof(DuLNode));
    p->next->prior=p;    p=p->next;
    p->data=x;
  }
  p->next=L;  L->prior =p;
}
```

2.3.4 双链表

(5) 双向链表输出

```
void PrintList_DuL(DuLinkList L){
  DuLinkList p;
  p=L->next;  printf("L->");
  while (p!=L){
    printf("%d->",p->data);
    p=p->next;
  }
  printf("\n");
}
```

2.3.4 双链表

```
//结点类型
typedef struct LNode { ElemType data; struct Node *next; }*Link,*Position;
//链表类型
typedef struct { Link head; Link tail; int size; } LinkList;
```

Status MakeNode(Link &p, ElemType e);
void FreeNode(Link &p);
Status InitList(LinkList &L) ;
Status DestroyList(LinkList &L);
Status ClearList(LinkList &L);
Status InsFirst(Link h, Link s);
Status DelFirst(Link h, Link &q) ;
Status Append (LinkList &L, Links);
Status Remove (LinkList &L, Lirik &q) ;
……

2.3.4 双链表

链表合并

```
Status MergeList_L(LinkList &La, LinkList &Lb, inkList &Lc,
                                        int(*compare)(ElemType,ElemType)){
//已知单链线性表a和b的元素按值非递减排列。归并a和得到新的单链线性表Lc,Lc的元素也按值非递减排列。
    if(!InitList(Lc))return ERROR;
    ha=GetHead(La);hb=GetHead(Lb);
    pa=NextPos(La,ha);pb=NextPos(Lb,hb);
    while(pa&&pb){
        a=GetCurElem(pa);b=GetCurElem(pb);
        if((*compare)(a,b)<0){// a<=b
            DelFirst(ha,q);Append(Lc,q);pa=NextPos(La,ha);
        }
        else{
            DelFirst(hb,q);Append(Lc,q);pb=NextPos(Lb,hb);
        }
    }
    if(pa) Append(Lc, pa);
    else Append(Lc,pb);
    FreeNode(ha); FreeNode(hb);
    return OK;
}
```

2.3.4 双链表

例：要求实现用户输入一个数字改变26个字母的排列顺序。正常情况下26个字母的排列顺序是A B C D E F G H I J K L M N O P Q R S T U V W X Y Z。当用户输入3时，字母的排列顺序改为D E F G H I J K L M N O P Q R S T U V W X Y Z A B C。当用户输入-3时，字母的排列顺序改为X Y Z A B C D E F G H I J K L M N O P Q R S T U V W 。

2.3.4 双链表

将26个英文字母排列成一个双向循环链表，头指针指向A，此时如果输入的数字是正数，则

```
for (int i = 0; i < num; i++){  //num代表输入的数字
    s = s->next;
}
```

输入的数字是负数，则

```
for (int i = 0; i < -num; i++){  //num代表输入的数字
    s = s->prior;
}
```

2.3.4 双链表

```
typedef struct Node{
    char c;
    struct Node* next;
    struct Node* prior;
}Node* List;


int main(void){
    List ok;
    int num;
    ok = createList();
    printf("请输入一个数字: ");
    scanf("%d", &num);
    mission(num, ok);
}
```

2.3.4 双链表

```
List createList(){
    List s, p;
    List Head;
    char c = 'A';
    int i;
    Head= (List)malloc(sizeof(struct Node));
    s = Head;                  //头结点节点初始化
    s->c = c;
    s->next = s;
    s->prior = s;
    for (i = 1; i < 26; i++){
        c = c + 1;
        p = (List)malloc(sizeof(struct Node));
        s->next = p;           //此处一共具有三个List节点，head始终指向头部
        p->prior = s;         //p节点是临时节点，用作每一次插入新节点。
        Head->prior = p;      //s节点每次都进行s=s->next更新，始终指向尾节点。
        p->next = Head;
        s = s->next;
        p->c = c;
    }
    return Head;
}
```

2.3.4 双链表

```
void ScanList(List list){
    List s = list;
    printf("%c ", s->c);     //从头到尾遍历链表
    s = s->next;
    while (s != list){
        printf("%c ", s->c);
        s = s->next;
    }
}
void mission(int num, List list){
    List s = list;
    if (num > 0)
    {
        for (int i = 0; i < num; i++){  //num代表输入的数字
            s = s->next;
        }
    }
    else{
        for (int i = 0; i < -num; i++){ //num代表输入的数字
            s = s->prior;
        }
    }
    ScanList(s);
}
```

第2章 线性表

- 2.1 线性表的类型定义
- 2.2 线性表的顺序表示和实现
- 2.3 线性表的链式表示和实现
- 2.4 一元多项式的表示及相加

2.4 一元多项式的表示及相加

一元多项式的表示

多项式的操作是表处理的典型用例。数学上,一元多项式可按降幂写成：（指数为正整数的情况）

$$P_n(x) = p_n x^n + p_{n-1} x^{n-1} + \dots p_1 x + p_0$$

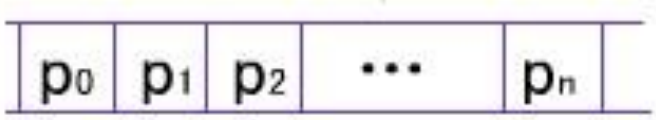
$$Q_m(x) = q_m x^m + q_{m-1} x^{m-1} + \dots q_1 x + q_0$$

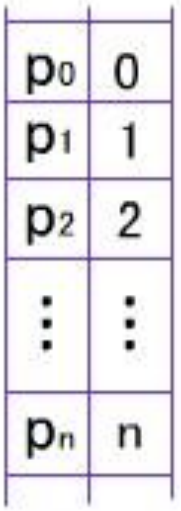
其中：p_n、q_m不为0

2.4 一元多项式的表示及相加

□ 顺序存储结构

(1)只存储各项的系数，存储位置下标对应其指数项
→ 适合于非零系数多的多项式





(2)系数与指数均存入顺序表
→适合于零项很多，且指数较大

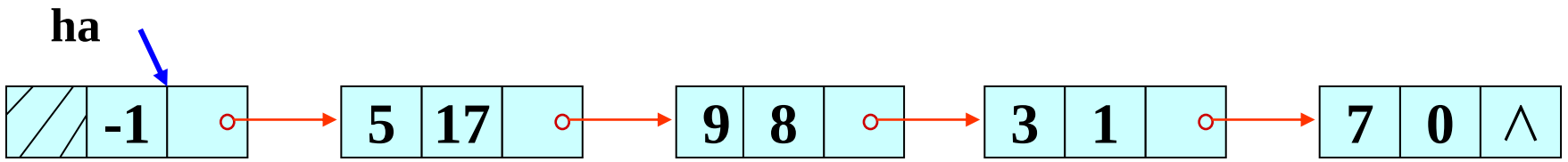
2.4 一元多项式的表示及相加

□ 链式存储结构

用线性链表表示。增加头结点，每个结点有

coef：系数 exp：指数 next：指针其中，头结点的exp为-1。

多项式 $A(x) = 5x^{17} + 9x^{8} + 3x + 7$



2.4 一元多项式的表示及相加

链式存储结构

可用单链表存储多项式的结点结构：

```
typedef struct Polynode
{
    int coef;
    int exp;
    Polynode *next;
} Polynode , *Polylist;
```

系数 coef　　指数 exp　　指针 next

2.4 一元多项式的表示及相加

输入多项式的系数和指数，用尾插法建立一元多项式的链表。

```
Polylist  polycreate() {
    Polynode *head, *rear, *s;
    int c,e;
    rear=head =(Polynode *)malloc(sizeof(Polynode));
    scanf("%d,%d",&c,&e);
    while(c!=0) {
        s=(Polynode*)malloc(sizeof(Polynode));
        s->coef=c;
        s->exp=e;
        rear->next=s;
        rear=s;
        scanf("%d,%d",&c,&e);
    }
    rear->next=NULL;
    return(head);
}
```

```
typedef struct Polynode
{
    int coef;
    int exp;
    Polynode *next;
} Polynode , *Polylist;
```

2.4 一元多项式的表示及相加

两个多项式相加

运算规则：两个多项式中所有指数相同的项的对应系数相加，若和不为零，则构成“和多项式”中的一项；所有指数不相同的项均复抄到“和多项式”中。

例：求两多项式的和多项式

$$A(x) = 5x^{17} + 9x^{8} + 3x + 7$$

$$B(x) = -9x^{8} + 22x^{7} + 8x$$

一元多项式相加运算规则：指数相同的项系数相加

A (x) B (x)相加的和多项式为

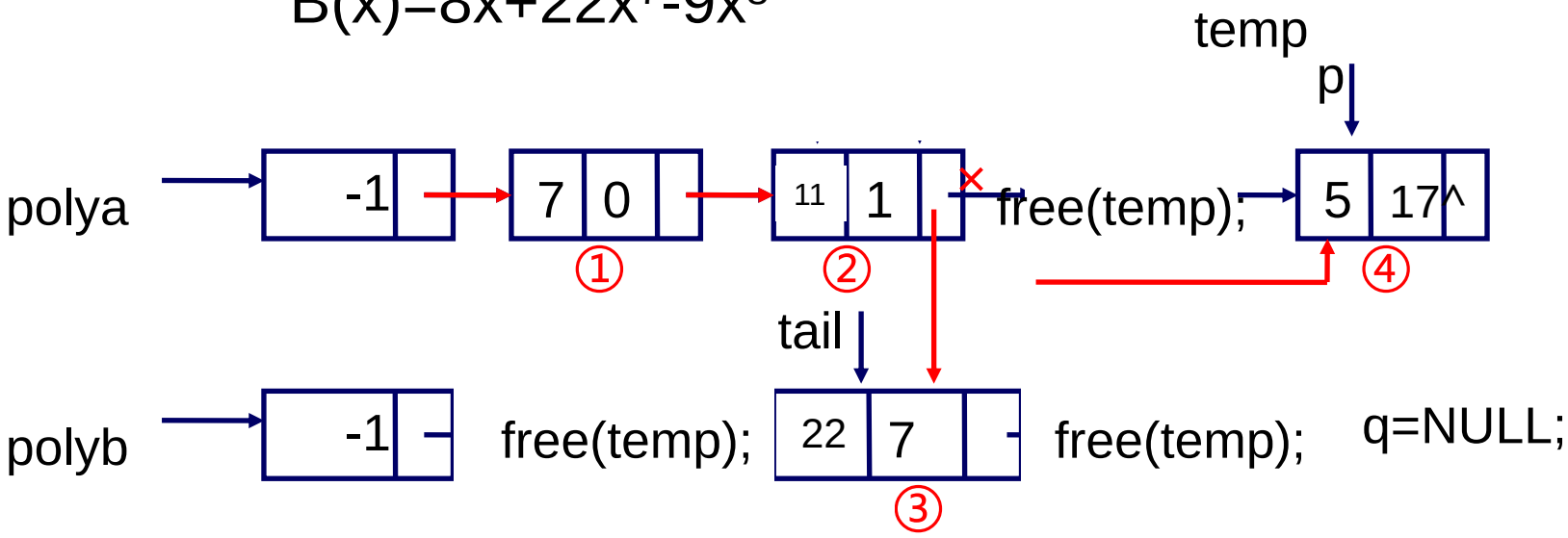
$$C(x) = A(x) + B(x) = 5x^{17} + 22x^{7} + 11x + 7$$

2.4 一元多项式的表示及相加

两个多项式相加

$A(x)=7+3x+9x^8+5x^{17}$

$B(x)=8x+22x^7-9x^8$



2.4 一元多项式的表示及相加

两个多项式相加

算法思想：

（1）若p->exp<q->exp，则结点p所指的结点应是“和多项式”中的一项，令指针p后移；

（2）若p->exp=q->exp，则将两个结点中的系数相加，当和不为零时修改结点p的系数域，释放q结点；

（3）若和为零，则和多项式中无此项，删去p和q结点，同时释放p和q结点;

（4）若p->exp>q->exp，则结点q所指的结点应是“和多项式”中的一项，将结点q插入在结点p之前，且令指针q在原来的链表上后移。

2.4 一元多项式的表示及相加

```
void  PolyAdd(Polylist polya, Polylist polyb){
    Polynode *rear, *p, *q, *temp;
    int sum;
    p=polya->next;   q=polyb->next;
    rear=polya;    free(polyb);
    while(p&&q){
        if(p->exp < q->exp)
        { rear->next=p;  rear=p;  p=p->next;}
        else if(p->exp > q->exp)
        { rear->next=q;  rear=q;  q=q->next;}
        else{
            sum=p->coef+q->coef;
            if(sum){
                p->coef=sum; rear->next=p; rear=p; p=p->next;
                temp=q; q=q->next; free(temp);   }
            else{
                temp=p;  p=p->next;  free(temp);
                temp=q;  q=q->next;  free(temp);  }
        }
    }
    if(p)  rear->next=p;  else  rear->next=q;
}
```

2.4 一元多项式的表示及相加

运算效率分析：

(1) 系数相加

0 ≤ 加法次数≤ min(m, n)

其中 m和n分别表示表A和表B的结点数。

(2)指数比较

极端情况是表A和表B 没有一项指数相同，比较次数最多为m+n-1

(3)新结点的创建

极端情况是产生m + n 个新结点合计时间复杂度为 O(m+n)

2.4 一元多项式的表示及相加

ADT Polynomial{

数据对象：$D=\{a_i|a_i \in TermSet, i=1,2,\ldots,m, m\geq 0$

TermSet中的每个元素包含一个表示系数的实数和表示指数的整数}

数据关系：$R1=\{<a_{i-1},a_i>|\ a_{i-1}, a_i \in D,$

且a_{i-1}中的指数值<a_i中的指数值, $i=1,2,\ldots,n\}$

基本操作：

线性链表若干公共基本操作的定义见教材P37（此处略）

为本例定义的更多基本操作有：

2.4 一元多项式的表示及相加

CreatPolyn(&P,m)

操作结果：输入m项的系数和指数，建立一元多项式P。//建表

DestroyPolyn(&P)　　//销毁也是P的一种变化

初始条件：一元多项式P已存在。

操作结果：销毁一元多项式P。　　//释放表

PrintPolyn(P)

初始条件：一元多项式P已存在。

操作结果：打印输出一元多项式P。　//输出一元多项式表

PolynLength(P)　　//项数=表长

初始条件：一元多项式P已存在。

操作结果：返回一元多项式P中的项数。//求表长，用函数值返回

2.4 一元多项式的表示及相加

AddPolyn(&P_a,&P_b)

初始条件：一元多项式P_a和P_b已存在。

操作结果：完成多项式相加运算，即：$P_a=P_a+P_b$，
并销毁一元多项式P_b。 //两表相加

SubtractPolyn(&P_a,&P_b)

初始条件：一元多项式P_a和P_b已存在。

操作结果：完成多项式相减运算，即：$P_a=P_a-P_b$，
并销毁一元多项式P_b。 //两表相减

MultiplyPolyn(&P_a,&P_b)

初始条件：一元多项式P_a和P_b已存在。

操作结果：完成多项式相乘运算，即：$P_a=P_a\times P_b$，
并销毁一元多项式P_b。 //两表相乘

}ADT Polynomial

2.4 一元多项式的表示及相加

求多项式$q(x)=3x^{14}-8x^{8}+6x^{2}+2$和$p(x)=2x^{10}+4x^{8}-6x^{2}$的乘积

先可将$q(x)$与$p(x)$的最高幂次项$2x^{10}$相乘，得$6x^{24}-16x^{18}+12x^{12}+4x^{10}$;

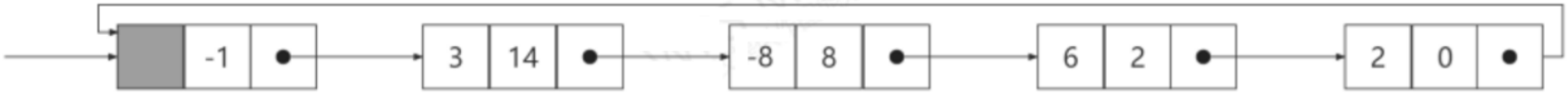
再将$q(x)$与$p(X)$下一项$4x^{8}$相乘，得$12x^{22}-32x^{16}+24x^{10}+8x^{8}$;

再将$q(x)$与$p(x)$的最后一项$-6x^{2}$相乘，得$-18x^{16}+48x^{10}-36x^{4}-12x^{2}$;

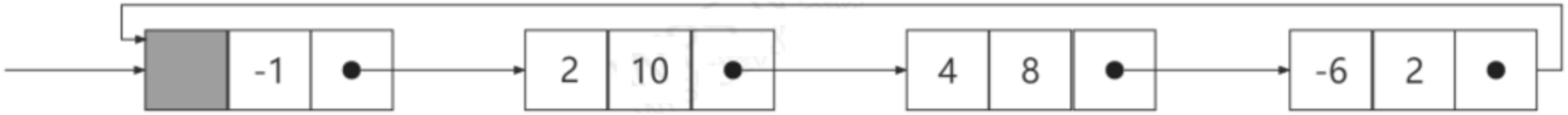
最后将这三个多项式相加便得到$q(x)\times p(x)$的乘积多项式：

$$6x^{24}+12x^{22}-16x^{18}-50x^{16}+12x^{12}+76x^{10}+8x^{8}-36x^{4}-12x^{2}$$

2.4 一元多项式的表示及相加



$$a(x) = 3x^{14} - 8x^{8} + 6x^{2} + 2$$



$$b(x) = 2x^{10} + 4x^{8} - 6x^{2}$$

$$6x^{24} + 12x^{22} - 16x^{18} - 50x^{16} + 12x^{12} + 76x^{10} + 8x^{8} - 36x^{4} - 12x^{2}$$

线性表知识要点

线性表知识要点

定义

线性表： 由$n(n\geqslant 0)$个数据元素(结点)a_1，a_2，…a_n组成的有限序列。其中数据元素的个数n定义为表的长度。当n=0时称为空表，常常将非空的线性表$(n>0)$记作：

$$(a_1，a_2，\ldots a_n)$$

其中a_i是属于某一个数据对象的数据元素。

特点

- 线性表中所有元素的性质是相同的，即具有相同数据类型。
- 在非空的线性表，有且仅有一个开始结点a_1，它没有直接前趋，而仅有一个直接后继a_2；
- 有且仅有一个终端结点a_n，它没有直接后继，而仅有一个直接前趋a_{n-1}；
- 其余的内部结点$a_i\ (2\leqslant i\leqslant n-1)$都有且仅有一个直接前趋$a_{i-1}$和一个直接后继$a_{i+1}$。

线性表知识要点

- 基本运算
  - 存取：存取或更新表中某个数据元素。
  - 插入：在表的两个确定的元素之间插入一个新元素。
  - 删除：删除表中某个数据元素。
  - 查找：查找表中满足某种条件的数据元素。如找出某个数据项具有给定值的数据元素。
- 复杂运算
  - 合并：把两个线性表合并成一个线性表。
  - 分解：把一个线性表拆分成多个线性表。
  - 排序：按一个或多个数据项值的递增或递减次序重新排列表中数据元素。

线性表知识要点

- 顺序存储 — —顺序表

把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里。

假设线性表的每个元素需占用L个存储单元，并以所占的第一个单元的存储地址作为数据元素的存储位置。

线性表中第i+1个数据元素的存储位置$LOC(a_{i+1})$：

$$LOC(a_{i+1})=LOC(a_1)+L*i$$

线性表的第i个数据元素a_i的存储位置为：

$$LOC(a_i)=LOC(a_1)+(i-1)*L$$

通常称$LOC(a_1)$为线性表的开始地址。

适合很少进行插入和删除，但要求以最快速度存取表中元素的情况。

线性表知识要点

- 顺序表静态存储结构

```
# define ListSize   100
  typedef  int  DataType;
  typedef  struct {
        DataType  data[ListSize];
        int   length;
  } Sqlist;
```

线性表知识要点

- 顺序表静态存储结构

```
# define LIST_INIT_SIZE     100
# define LISTINCREMENT    10
   typedef   int  DataType ;
   typedef  struct {
       DataType *data ;
       int   length;
       int   listsize;
   } Sqlist;
```

线性表知识要点

顺序表——插入

- 掌握顺序表插入（向后移动数据），删除算法（向前移动数据），查找运算。

```
i--;  //转为物理序号
for (j=L->length; j>i; j--)
     L->data[j]=L->data[j-1]; //将data[i]及后面元素后移一个位置
L->data[i]=e;
L->length++;
```

```
 i--;
e=L->data[i];
for (j=i; j<L->length-1; j++)
     L->data[j]=L->data[j+1]; //将data[i]之后的元素后前移一个位置
L->length--;
```

线性表知识要点

链式存储 — —链表

链表是指用一组任意的存储单元来依次存放线性表的结点，这组存储单元即可以是连续的，也可以是不连续。

链式存储结构，简称为链表(Linked List)。

- 单链表
- 双向链表
- 循环单链表
- 循环双链表
- 静态链表

线性表知识要点

- 链式存储 — 一链表

- 单链表、循环单链表

```
typedef struct LNode {
    ElemType      data;     //数据域
    struct Lnode  *next;    //指针域
}LNode, *LinkList;
```

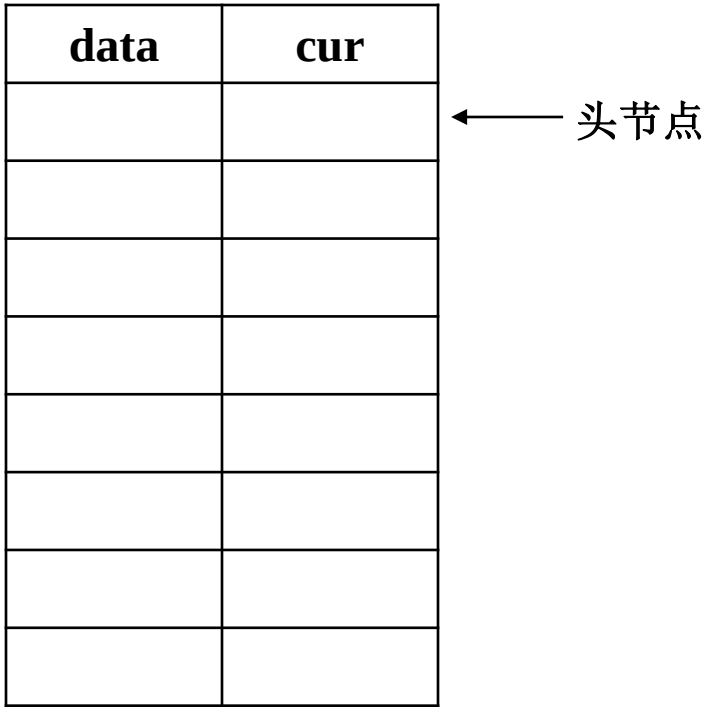
- 双向链表、循环双链表

```
Typedef struct DuLNode{
    ElemType        data;       //数据域
    struct DuLNode  *prior;     //前驱指针域
    struct DuLNode  *next;      //后继指针域
} DuLNode , *DuLinkList ;
```

线性表知识要点

- 链式存储 — —链表
  - 静态链表

```
#define MAXSIZE   1000   //预分配最大的元素个数
typedef   struct  {
      ElemType   data ;    //数据域
      int              cur ;    //指示域
}component , SLinkList[MAXSIZE] ;
```



线性表知识要点

链式存储——链表——创建链表

- **头插法**

```
s = (LNode *)malloc(sizeof(LNode));
s->data = a[i];
s->next = L->next;        s->prior = L;
L->next = s;
```

- **尾插法**

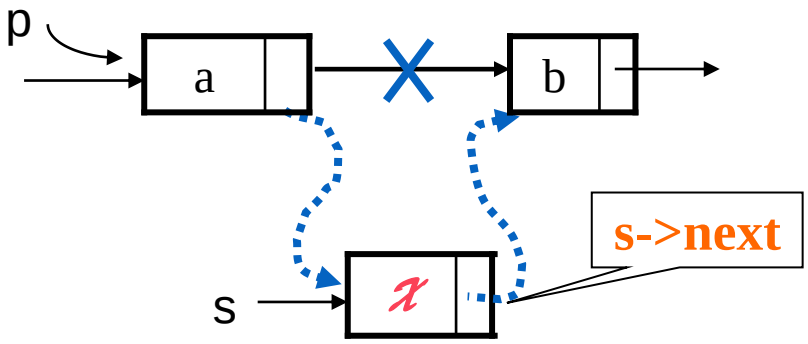
```
L = (LNode *)malloc(sizeof(LNode));
r = L;
s = (LNode *)malloc(sizeof(LNode));
s->data = a[i];
r->next = s;                s->prior = r;
r = s;
```

线性表知识要点

链式存储——链表——插入

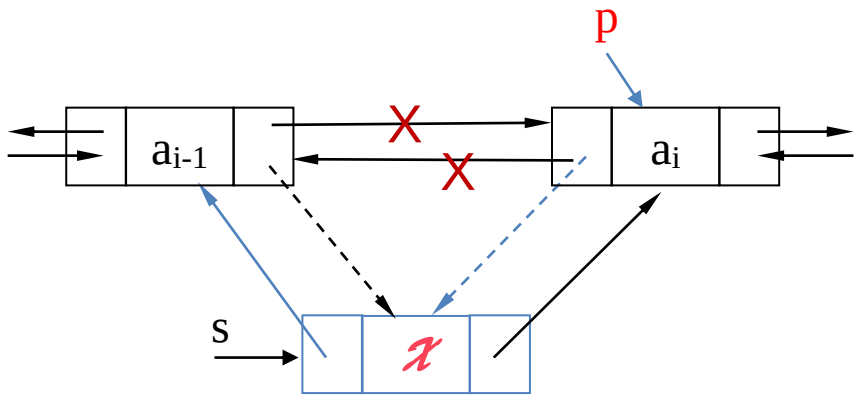
- 单链表

```
s=(LNode*)malloc(sizeof(LNode));
s->data=x;
s->next=p->next;
p->next=s;
```



- 双向链表

```
s=(LNode*)malloc(sizeof(LNode));
s->data=x;
s->next = p;
p->prior->next = s ;
s->prior = p ->prior ;
p->prior = s ;
```

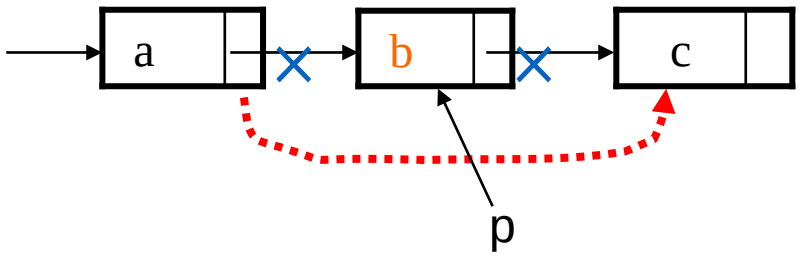


线性表知识要点

链式存储——链表——删除

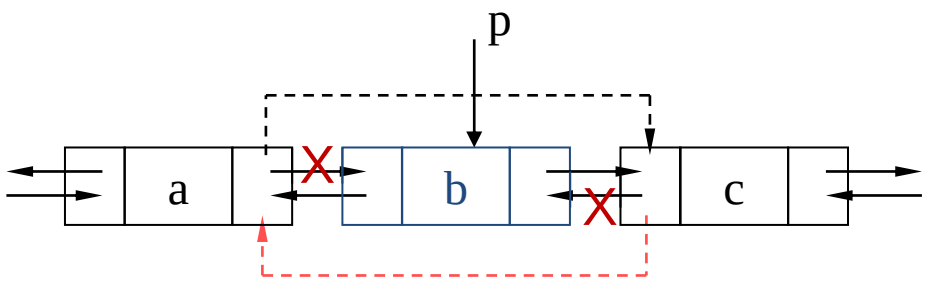
- 单链表

```
p->prior->next = p->next;
free(p);
```



- 双向链表

```
p->prior->next = p->next;
p->next->prior = p->prior;
free(p);
```



正在答疑