

# 数据结构

深圳技术大学

# 第二章 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 一元多项式的表示和实现**

# 复习

⑩ 线性表： $n$ 个数据元素的有限序列

⑩ 线性表顺序表示：用一组地址连续的存储单元依次存储线性表的数据元素

- ❧ 随机存取

- ❧ 增删时需要移动大量元素

⑩ 顺序表链式表示：存储单元并不连续，通过指针等手段来表示数据之间的邻接关系

- ❧ 不可随机存取

- ❧ 增删时只需要修改前后节点

## B. DS顺序表之循环移位

### 题目描述

顺序表的移位是循环移位，例如顺序表：1，2，3，4，5，6。如果左移1位，即原来的头元素移动到末尾，其它元素向左移1位，变成2，3，4，5，6，1。同理，如果右移1位，即原来的尾元素移动到头，其它元素向右移1位，变成6，1，2，3，4，5。

以下是移位的多个例子：

原数据：1，2，3，4，5，6

左移3位：4，5，6，1，2，3，与原数据对比

右移4位：3，4，5，6，1，2，与原数据对比

请编写程序实现顺序表的循环移位操作

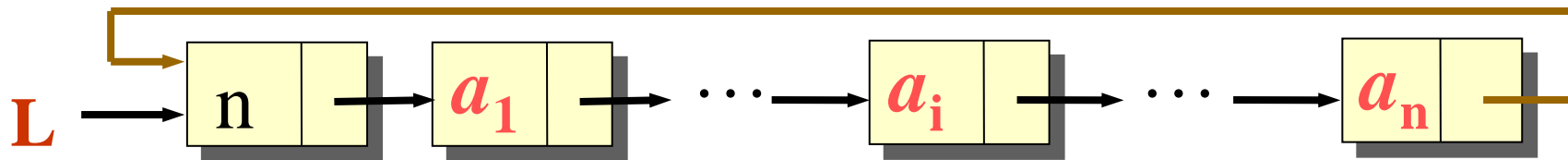
## 用单链表该怎么做？

## 如何在单链表中做首尾的循环？

## 2.3 链表

### 五. 循环链表

- 循环链表是一种特殊的线性链表
- 循环链表中最后一个结点的指针域指向头结点，整个链表形成一个环
- 循环链表的查找、插入、删除和单链表基本一致
- 与单链表的区别
  - (1) 判断是否是空链表:  $\text{head} \rightarrow \text{next} == \text{head}$  ;
  - (2) 判断是否是表尾结点:  $\text{p} \rightarrow \text{next} == \text{head}$  ;



如何输出线性表的逆序？

## 2.3 链表

### 六. 双向链表

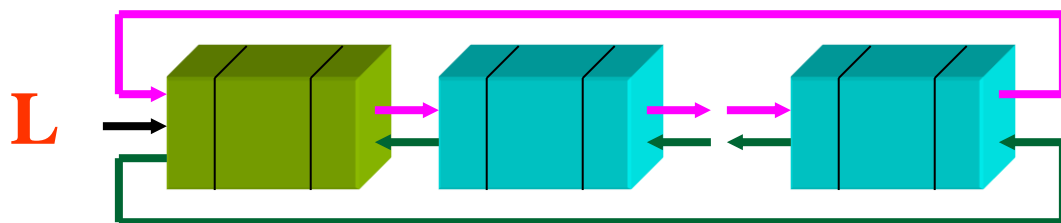
- 双向链表也是一种特殊的线性链表
- 双向链表中每个结点有两个指针，一个指针指向直接后继(next)，另一个指向直接前驱(prior)



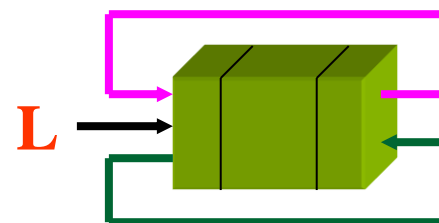
## 2.3 链表

### 六. 双向链表

- 双向链表中存在两个环(一个是直接后继环(红), 另一个是直接前驱环(蓝))



非空表



空表

## 2.3 链表

### 六. 双向链表

#### ■ 双向链表的定义

//定义一个双向链表的结点

```
class DuLNode {  
    ElemType data;  
    DuLNode *prior;  
    DuLNode *next;  
}
```

```
class DuLinkList {  
    DuLNode head;  
    ... .. // 其他数据成员  
public:  
    ... .. // 成员函数  
}
```

#### ■ 对于任何一个中间结点有:

- $p = p \rightarrow \text{next} \rightarrow \text{prior}$
- $p = p \rightarrow \text{prior} \rightarrow \text{next}$





## 2.3 链表

### 六. 双向链表

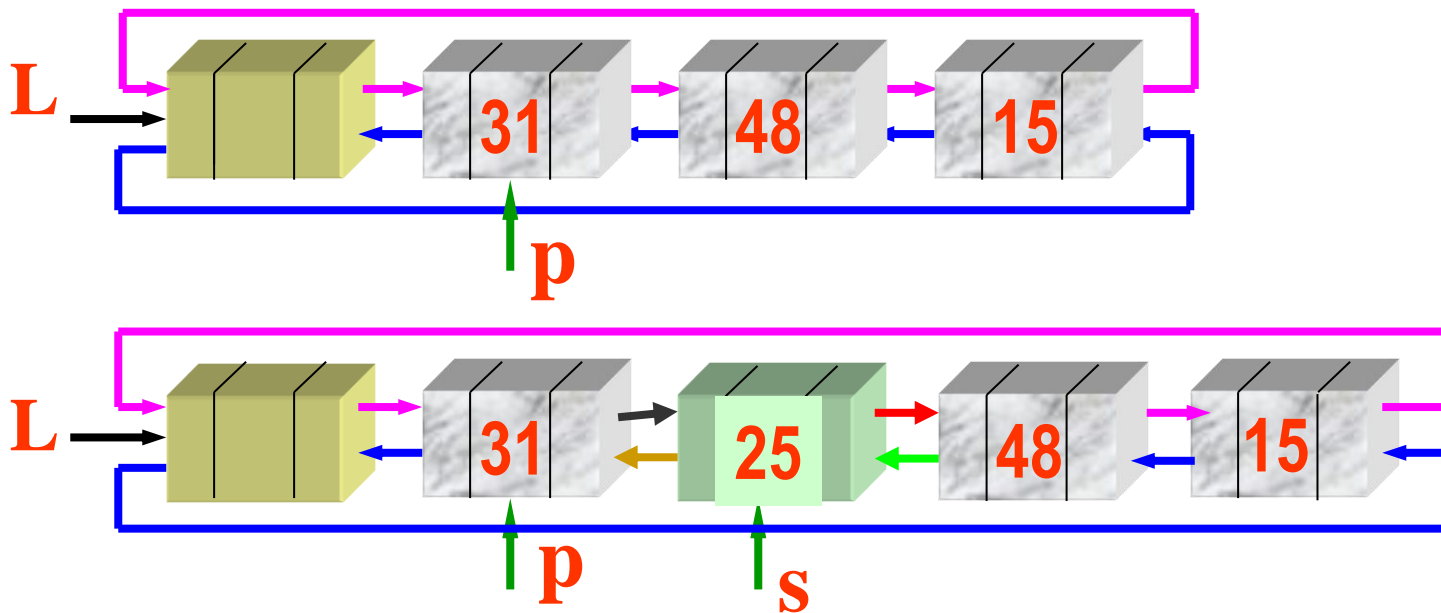
■ 双向链表的插入需要改变两个方向的指针

`s->next = p->next;`

`s->prior = p;`

`p->next->prior = s;`

`p->next = s;`



## 2.3 链表

### 六. 双向链表

#### ■ 双向链表的插入

```
Status DuLinkList<ElemType>::ListInsert_DuL(int i, ElemType e)
{
    // 在带头结点的单链表L中第i个位置插入元素e
    DuLNode<ElemType> *p = head->next;
    int j = 0;
    while (p!=&head && j<i-1) { p = p->next; j++; }           //
寻找i-1结点
    if (j!=i-1) return ERROR;
    DuLNode *s = new DuLNode<ElemType>;                       // 生成新结点
    s->data = e;
    s->next = p->next; s->prior = p;                             // 将指定数据插入到i-1的
next位置
    p->next->prior = s; p->next = s;                             // 修改前后结点的指针值
    return OK;
} // ListInsert_L
```

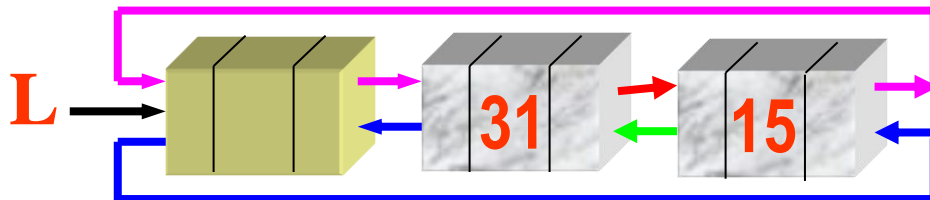
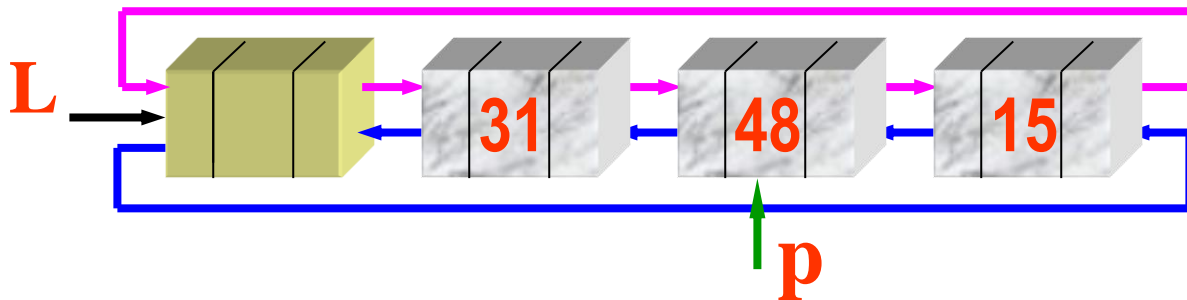
## 2.3 链表

### 六. 双向链表

■ 双向链表的删除需要改变两个方向的指针

$p \rightarrow \text{prior} \rightarrow \text{next} = p \rightarrow \text{next};$

$p \rightarrow \text{next} \rightarrow \text{prior} = p \rightarrow \text{prior};$



## 2.3 链表

### 六. 双向链表

#### ■ 双向链表的删除

```
template<class ElemType>
Status DuLinkList<ElemType>::ListDelete_L(int i, ElemType &e)
{
    // 在带头结点的单链表L中，删除第i个位置的元素
    DuLNode<ElemType> *p = head.next;
    int j = 0;
    while (p!=&head && j<i) { p = p->next; j++; } // 寻找i-1
    结点
    if (j<i) return ERROR;
    e = p->data;
    p->next->prior = p->prior;
    p->prior->next = p->next; // 删除i结点
    delete p;                // 释放结点
    return OK;
} // ListDelete_L
```

## 2.3 链表

### 六. 双向链表

#### ■ 双向链表获取第i个位置的元素

- L为带头结点的单链表头指针，当第i个元素存在时，其值赋给e并返回**OK**，否则返回**ERROR**

```
template<class ElemType>
DuLinkList<ElemType> GetElemP_DuL(int i) {
    int j = 0;    // 初始化，p指向第一个结点，j为计数器
    DuLNode<ElemType> *p = head.next;
    while (p!=&head && j<i) {
        // 顺指针向后查找，直到p指向第i个元素或p为空
        p = p->next;
        ++j;
    }
    if (j<i) return NULL;    // 第i个元素不存在
    else return p;
} // GetElem_L
```

## 2.3 链表

### 七. 顺序表与链表的比较（空间）

#### ■ 存储分配的方式

- 顺序表的存储空间是静态分配的
- 链表的存储空间是动态分配的

#### ■ 存储密度 = 结点数据本身所占的存储量/结点结构所占的存储总量

- 顺序表的存储密度 = 1
- 链表的存储密度 < 1

## 2.3 链表

### 七. 顺序表与链表的比较（时间）

#### ■ 存取方式

- ☐ 顺序表可以随机存取，也可以顺序存取
- ☐ 链表必须顺序存取

#### ■ 插入/删除时移动元素个数

- ☐ 顺序表平均需要移动近一半元素
- ☐ 链表不需要移动元素，只需要修改指针

## 2.3 链表

### 七. 顺序表与链表的比较（应用）

- 如果线性表主要是存储大量的数据，并主要用于查找时，采用顺序表较好，如数据库
- 如果线性表存储的数据元素经常需要做插入与删除操作，则采用链表较好，如操作系统中进程控制块(PCB)的管理，内存空间的管理等



# 思考题

习题1. 已知非空循环链表，设**h**是指向头结点的指针，**p**是辅助指针。执行以下程序段的作用是什么？

```
p=h;  
while (p->next->next!=h)  
    p=p->next;  
p->next=h;
```

习题2. 双向链表中前驱指针为**prior**，后继指针为**next**，在指针**P**所指结点前插入指针**S**所指的结点，请写出要执行的四行语句？

## 2.3 链表

### 八. 一元多项式的表示与运算

- 一元多项式  $p(x)=p_0+p_1x+p_2x^2+ \dots +p_nx^n$  ,
  - 由 $n+1$ 个系数唯一确定

**[例]** 一元多项式及其运算。

一元多项式：  $f(x) = a_0 + a_1x + \cdots + a_{n-1}x^{n-1} + a_nx^n$

主要运算：多项式相加、相减、相乘等

**【分析】** 多项式的关键数据是：多项式项数 **n**、每一项的系数 **a<sub>i</sub>**（及相应指数 **i**）。有3种不同的方法。

**方法1：** 采用顺序存储结构直接表示

例如：  $f(x) = 4x^5 - 3x^2 + 1$

表示成：

下标 <b>i</b>	<b>0</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	.....
<b>a[i]</b>	<b>1</b>	<b>0</b>	<b>-3</b>	<b>0</b>	<b>0</b>	<b>4</b>	.....

方法2：采用顺序存储结构表示多项式的非零项。

每个非零项  $a_i x^i$  涉及两个信息：指数  $i$  和系数  $a_i$ ，  
可以将一个多项式看成是一个  $(a_i, i)$  二元组的集合。

例如：  $P_1(x) = 9x^{12} + 15x^8 + 3x^2$  和  $P_2(x) = 26x^{19} - 4x^8 - 13x^6 + 82$

表示成：

数组下标 <i>i</i>	0	1	2	.....
系数	9	15	3	—
指数	12	8	2	—

(a)  $P_1(x)$

数组下标 <i>i</i>	0	1	2	3	.....
系数	26	-4	-13	82	—
指数	19	8	6	0	—

(b)  $P_2(x)$

方法3：采用链表结构来存储多项式的非零项。

每个链表结点存储多项式中的一个非零项，包括系数和指数两个数据域以及一个指针域，表示为：

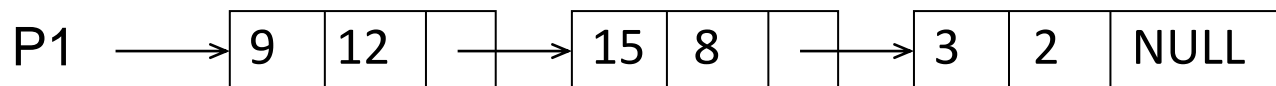
coef	expon	link
------	-------	------

例如：

$$P_1(x) = 9x^{12} + 15x^8 + 3x^2$$

$$P_2(x) = 26x^{19} - 4x^8 - 13x^6 + 82$$

链表存储形式为：



## 2.3 链表

### 八. 一元多项式的表示与运算

- 一元多项式  $p(x)=p_0+p_1x+p_2x^2+ \dots +p_nx^n$  ,
  - 由n+1个系数唯一确定
  - 在计算机中可用线性表( $p_0, p_1, p_2, \dots, p_n$ )表示。
  - 用顺序表和链表来实现。两种不同实现方式的元素类型定义如下:

#### (1) 顺序存储表示的类型

```
typedef struct{  
    /*系数部分*/  
    float    coef;  
    /*指数部分*/  
    int      expn;  
} ElemType ;
```

#### (2) 链式存储表示的类型

```
class poly{  
    //系数部分  
    float coef;  
    //指数部分  
    int    expn;  
    //指向下一个结点  
    poly* next;  
    ... // 其他成员  
};
```

## 2.3 链表

### 八. 一元多项式的表示与运算

#### ■ 一元多项式的相加

不失一般性，设有两个一元多项式：

$$P(x)=p_0+p_1x+p_2x^2+ \dots +p_nx^n ,$$

$$Q(x)=q_0+q_1x+q_2x^2+ \dots +q_mx^m \quad (m<n)$$

$$R(x)=P(x)+ Q(x)$$

$R(x)$ 由线性表 $R((p_0+q_0), (p_1+q_1), (p_2+q_2), \dots, (p_m+q_m), \dots, p_n)$ 唯一表示。

## 2.3 链表

### 八. 一元多项式的表示与运算

#### ■ 顺序存储表示的相加

- 用顺序表示的相加非常简单。访问第5项可直接访问：**L.a[4].coef** , **L.a[4].expn**
- 两个多项式相加就是在两个顺序表中寻找指数**expn**相同的元素把两者的系数**coef**相加
- 例如 **$5+x+2x^2+3x^3$** ,  **$-5-x+6x^2+4x^4$** , 两者相加
  - X的0次方的系数为0
  - X的1次方的系数为0
  - X的2次方的系数为8
  - X的3次方的系数为3
  - X的4次方的系数为4
  - 系数为0的部分不显示, 最终结果:  **$8x^2+3x^3+4x^4$**
- 相关顺序表操作: 按值查找、插入、合并



## 2.3 链表

### 八. 一元多项式的表示与运算

#### ■ 链式存储表示的相加

- 当采用链式存储表示时，根据结点类型定义，凡是系数为**0**的项不在链表中出现，从而可以大大减少链表的长度。

#### ■ 相加的实质是：

- 指数不同： 是链表的合并
- 指数相同： 系数相加，和为**0**，去掉结点，和不为**0**，修改结点的系数域

#### ■ 程序实现的操作包括：

- 多项式链表创建、相加、输出
- 项插入、删除、查找

friend poly **operator+**(poly &La, poly &Lb) { //以运算符重载的方式实现, La和Lb为两个系数链表的头结点

poly Lc; // 实例化另一个头结点保留最终结果

poly \*pa = &La; poly \*pb = &Lb; poly \*pc = &Lc;

\*pa = La->next; pb = Lb->next; // 开始循环计算

**while** (pa!=NULL && pb!=NULL) {

**if** (pa->expn < pb->expn) {

        pc->next = pa; pc = pa; pa = pa->next;

    } **else if** (pa->expn > pb->expn) {

        pc->next = pb; pc = pb; pb = pb->next;

    } **else** {

        float x = pa->coef + pb->coef ;

**if** (abs(x) <= 1.0e-6) { //如果系数和为0, 删除两个结点

            poly \*ptrb = pa; pa = pa->next; **delete** ptrb;

            poly \*ptrb = pb; pb = pb->next; **delete** ptrb;

        } **else** { // 如果系数和不为0

            pc->next = pa; pa->coef = x;

            pc = pa; pa = pa->next;

            poly \*ptrb = pb; pb = pb->next; **delete** ptrb;

        }

    } //end else

} // end while

**if** (pa == NULL) pc->next=pb; **else** pc->next=pa ;

**return** Lc ;

# Take Home Message

⑩ 双向链表：使用**prior**和**next**指针表示邻接关系

⑩ 链表的优缺点：

❧ 缺点：无法进行随机存取

❧ 优点：相较于顺序表，能够更快地进行增删操作

⑩ 一元 $n$ 次方程的表示与计算

❧ 稀疏顺序表，稠密顺序表，链表

❧ 一元 $n$ 次方程的加法操作：类似线性表的合并操作

## 2.3 链表

思考题：

### ■ 【问题描述】

13 张黑桃扑克（A 2 3 4 5 6 7 8 9 10 J Q K），预先排好，正面朝下拿在魔术师的手里，从最上面开始，第一次数一张牌翻过来放在桌面上，正好是“A”；第二次数两张牌，数1的那张放在手中扑克的最下面，数2的那张翻过来放在桌面上正好是“2”；……,如此下去，放在桌面上的牌最后正好是“A 2 3 4 5 6 7 8 9 10 J Q K”的顺序（从下向上）。

【任务】编程找出魔术师手中扑克原来的排列顺序（从下向上）。

### ■ 【问题描述】

围绕着山顶有10个洞，一只兔子和一只狐狸各住一个洞，狐狸总想吃掉兔子。一天兔子对狐狸说，你想吃我有一个条件，第一次隔一个洞找我，第二次隔两个洞找我，以后依次类推，次数不限。若能找到我，你就可以饱餐一顿，在没找到我之前不能停止。狐狸一想只有10个洞，寻找的次数又不限，哪有找不到的道理，就答应了条件。结果就是没找着。现请你编写一程序，假定狐狸找了1000次，兔子躲在哪个洞里才安全。