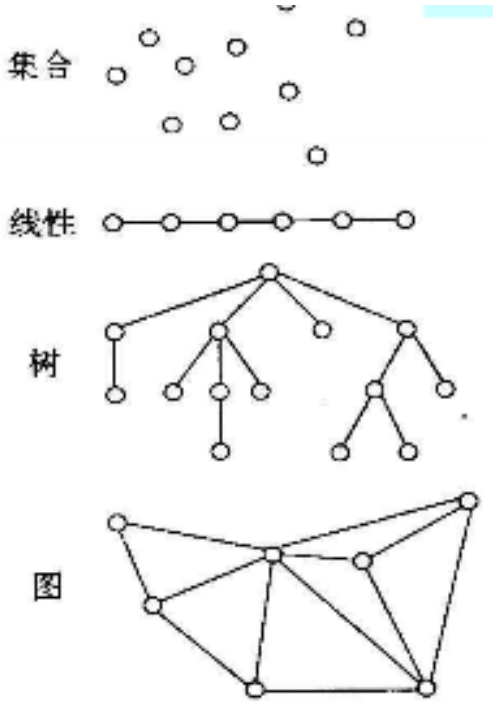
第2章 线性表

——刘亮亮

上海对外经贸大学
SHANGHAI UNIVERSITY OF INTERNATIONAL BUSINESS AND ECONOMICS

回顾...

- **上节课的主要内容：**
  - 数据结构的定义
  - 数据结构的三要素：
    - ✓逻辑结构
    - ✓存储结构
    - ✓操作
  - 四种数据结构
  - 抽象数据类型ADT={D,R,P}
  - 算法与算法分析
    - ✓定义与五大特性
    - ✓时间复杂度与空间复杂度



本章要点

- 线性表的定义与表示
- 线性表的顺序存储结构——顺序表
- 顺序表的操作与实现
- 线性表的链式存储结构——链表
- 链表的操作与实现

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

线性表的定义

- **线性结构：数据元素的非空有限集合。**
  - 存在唯一的一个称为“第一个”的数据元素
  - 存在唯一的一个称为“最后一个”的数据元素
  - 除第一个外，每一个元素有且只有一个前驱
  - 除最后一个外，每一个元素有且只有一个后继
- **线性表：是一种线性结构，n个数据元素的有限序列。**
- **例如:**
  - (A,B,C,.....Z)
  - (10,20,39,50,89)
  - 学生登记表

线性表的定义

- **一般地，将线性表表示为:**

$$List=(a_1,\ldots,a_{i-1},a_i,\ldots,a_n)$$

  - n称为线性表的表长。
  - 空表：n=0
  - a_{i-1}是a_i的直接前驱，a_i是a_{i-1}的直接后继
  - $a_i(1<=i<=n)$是相同类型的数据元素。
- **线性表是一种非常简单、常用、灵活的线性结构。**

线性表的定义

- **抽象数据类型定义**

```
ADT List{
    数据对象：D={a_i| a_i∈ElemSet,i=1,2,...,n}
    数据关系：R1={<a_i-1,a_i>| a_i-1∈ D,a_i∈D,i=2,3...n}
    基本操作：
        InitList(&L)
            操作结果：构造一个空的线性表L
        DestroyList(&L)
            初始条件：线性表 L 已存在
            操作结果：销毁线性表 L
      . . .
      GetElem(L,i,&e)
            初始条件：线性表L已存在，1<=i<=ListLength(L)
            操作结果：用e返回L中第i个数据元素的值
      LocateElem(L,e,compare())
      PriorElem(L,cur_e,&pre_e)
      NextElem(L,cur_e,&next_e)
      ListInsert(&L,i,e)
      ListDelete(&L,i ,&e)
      ..................
}
```

线性表的定义

- **基本操作：**
  - 构造Init:构造空的线性表
  - 销毁Destroy：销毁线性表
  - 清空Clear:将表置为空表
  - 求长度Length:求线性表的长度
  - 访问Locate：访问线性表中的指定的数据元素的位序
  - 取元素Get:取线性表中的第i个元素
  - 插入Insert：在线性表中插入一个新的数据元素
  - 删除Delete：在线性表中删除指定的数据元素
  - 求前驱Prior:求指定元素的前驱
  - 求后继next:求指定元素的后继

线性表的定义

- **示例1：利用线性表基本操作实现两个集合A和B的合并**
  - 分析：
    - ✓将存在于集合LB中的元素但不存在于线性表LA中的元素插入到线性表LA中
  - 算法思想：
    - ✓ Step 1:分别求得LA、LB的长度
      - lengthA=ListLength(La);
      - lengthB=ListLength(Lb);
    - ✓ Step 2:遍历线性表LB
      - for(int i=1;i<=lengthB;i++)
    - ✓ Step 3:依次取出LB中的元素b_i
      - Get(Lb,i,e)
    - ✓ Step 4:判断b_i是否LA中的元素
      LocateElem(La,e,equal))
      - 如果不是，插入到LA的末尾
      Insert(La,++lengthA,e)
      - 如果是，不插入
    - ✓ Step 5:重复Step3~Step4

算法描述：

```
void  union(List& La, List Lb){
   //将所有在线性表Lb中不在La中的元素插入La中
   //求La的长度
    int lengthA=ListLength(La);
   //求Lb的长度
    int lengthB=ListLength(Lb);
    //遍历线性表Lb
    for(int i=1;i<lengthB;i++){
        //取Lb中的第i个元素
        GetElem(Lb,i,e);
        //判断e是否La中
        if(!LocateElem(La,e,equal)){
            //插入到La的末尾
             Insert(La.++lengthA,e);
        }
    }
}//end union
```

线性表的定义

- **示例2：两个有序表的归并**
  - 分析：建立一空表Lc，依次取出La和Lb中的元素，比较大小，插入到Lc。
  - 算法思想：
    - ✓ Step 1:设定指针i,j分别指向La和Lb的第一个元素
    - ✓ Step 2:分别取出La的第i个元素a_i，Lb的第j个元素b_j
    - ✓ Step 3:比较a_i和b_j
      - 若ai<=bj,插入ai,指针i后移
      - 若a_i>b_j,插入b_j,指针j后移
    - ✓ Step 4:重复Step2~Step3,直到指针i或j到达表末尾
    - ✓ Step 5:如果指针i未到La的末尾，将La中剩余的插入
    - ✓ Step 6:如果指针j未到Lb的末尾，将Lb中剩余的插入

```
i=j=1;
k=0;
```

```
GetElem(La,i,ai);
GetElem(Lb,j,bi);
```

```
ListInsert(Lc,++k,ai);
i++;
```

```
ListInsert(Lc,++k,bj);
j++;
```

```
i<=lenA&&j<=lenB
```

```
while(i<=lenA)
{
  GetElem(La,i++,ai);
  Insert(Lc,++k,ai);
}
```

```
while(j<=lenB)
{
  GetElem(Lb,j++,bj);
   Insert(Lc,++k,bj);
}
```

上海对外经贸大学

线性表的定义

- **示例2：两个有序表的归并**

算法描述：

```
void  Merge(List La, List Lb,List &Lc){
    InitList(Lc);
    i=j=1;
    k=0;
   int lenA=ListLength(La);
    int lenB=ListLength(Lb);
    while((i<=lenA)&&(j<=lenB)){
        GetElem(La,i,ai);
        GetElem(Lb,j,bj);
        if(ai<=bj){
          Insert(Lc,++k,ai);
          ++i;
        }
        else{
          Insert(Lc,++k,bj);
          ++j;
        }
    }
   while(i<=lenA){//插入La中剩下的元素
       GetElem(La,i++,ai);
        Insert(Lc,++k,ai)
    }
   while(j<=lenB){//插入Lb中剩下的元素
       GetElem(Lb,j++,bj);
        Insert(Lc,++k,bj)
    }
}//end Merge
```

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

顺序表的表示与实现

- **线性表的顺序表示：用一组地址连续的存储单元依次存储线性表中的数据元素。**
- **例如：每个元素占L个存储单元**
  - $Loc(a_{i+1})=Loc(a_i)+L$
  - $Loc(a_i)=Loc(a_1)+(i-1)*L$

元素	地址
a_1	b
a_2	b+L
…	…
a_i	b+(i-1)*L
….	
a_n	b+(n-1)*L
…	

- **顺序表：**
  - 利用顺序存储结构存放的线性表

顺序表的表示与实现

- **顺序表的描述**

```
#define LIST_INIT_SIZE 100  //初始分配容量
#define LISTINCREMENT 10 //空间分配增量
Typedef struct{
    ElemType *elem;//存储空间基址
    int  length;  //当前长度
    int listsize;  //当前分配的存储容量
}SqList;
```

```
//线性表的初始化算法
Status Init(SqList &L){
    //构造空的线性表

L.elem=(ElemType*)malloc(LIST_INIT_SIZ
E*sizeof(ElemType));
if(!L.elem)exit(OVERFLOW);
 L.length=0;
 L.listsize=LIST_INIT_SIZE;
 return OK;
}
```

顺序表的表示与实现

- **顺序表的操作**
  - 操作1：插入操作——在第i个元素前面插入一个新的元素。
    - ✓算法思想：
      - Step 1:判断插入位置是否合法
      i<1||i>L.length+1
      - Step 2:判断空间是否满，如果满则重新分配
      L.length>=L.listsize
      - Step 3:将a_n~a_i依次向后移动一个位置
      q=&(L.elem[i-1]);//保存q为插入位置
      for(L.elem[l.length-1];p>=q;--p)
      ***(p+1)=*p;**
      - Step 4:在原a_i的位置插入元素b
      ***q=e;**
      - Step 5:表长增1
      ++L.length;

插入前	插入后
a_1	a_1
a_2	a_2
…	…
a_{i-1}	a_{i-1}
a_i	b
….	a_i
a_n	….
	a_n

顺序表的表示与实现

- **顺序表的操作**
  - 操作1：插入操作——在第i个元素前面插入一个新的元素。
    - ✓算法描述

```
Status Insert(SqList &L,int i,ElemType e){
    //插入位置是否合法
     if(i<1||i>L.length+1){
         return ERROR;
     }
    //存储空间是否已满，已满则重新分配
     if(L.length>=L.listsize){
        newbase=(ElemType *)realloc(L.elem, (L.listsize+LISTINCREMENT)    *sizeof(ElemType));
     }
     if(!newbase){  //分配失败
        exit(OVERFLOW);
     }
      q=&(L.elem[i-1]);    //保存当前插入位置
      for(p=&(L.elem[L.length-1]);p>=q;--p){  //移动元素
        *(p+1)=*p;
       }
       *q=e;  //插入e
       ++L.length; //表长增1
       return OK;
}
```

顺序表的表示与实现

- **顺序表的操作**
  - 操作1：插入操作——在第i个元素前面插入一个新的元素。
    - ✓算法分析

移动次数=n-i+1

假如在第i个元素前插入一个元素的概率为p_i

则平均移动次数

$$E = \sum_{i=1}^{n+1} p_i(n - i + 1)$$

若在任何位置上插入都是等概率事件，即$p_i=1/(n+1)$

则平均移动次数E重写为

$$E = \frac{1}{n+1}\sum_{i=1}^{n+1}(n - i + 1) = \frac{n}{2}$$

因此，插入算法的时间复杂度为O(n)

顺序表的表示与实现

- **顺序表的操作**
  - 操作2：删除操作——删除第i个元素
    - ✓算法思想
      - □Step 1:判断删除位置是否合法

        i<1||i>L.length
      - □Step 2:找到删除的位置和元素

        p=&(L.elem[i-1]);

        e=*p;
      - □Step 3:将a_{i+1}~a_n依次向前移动一个位置

        q=L.elem+L.length-1;

        for(++p;p<=q;++p) *(p-1)=*p;
      - □Step 4:表长减1

        --L.length;

删除前	删除后
a_1	a_1
a_2	a_2
…	…
a_{i-1}	a_{i-1}
a_i	a_{i+1}
….	…
a_{n-1}	a_n
a_n	

顺序表的表示与实现

- **顺序表的操作**
  - 操作1：删除操作——删除第i个元素
    - ✓算法描述

```
Status Delete(SqList &L,int i,ElemType& e){
     //删除位置是否合法
      if(i<1||i>L.length){
          return ERROR;
      }
      p=&(L.elem[i-1]);    //当前删除位置
      e=*p;
      //表尾位置
      q=L.elem+L.length-1;
       for(++p;p<=q;++p）{//移动元素
          *(p-1)=*p;
        }
        --L.length; //表长减1
        return OK;
}
```

顺序表的表示与实现

- **顺序表的操作**
  - 操作2：删除操作——删除第i个元素
    - ✓算法分析

移动次数=n-i

假如在删除元素的概率为q_i

则平均移动次数

$$E = \sum_{i=1}^{n} q_i(n - i)$$

若在任何位置上删除都是等概率事件，即$q_i=1/n$

则平均移动次数E重写为

$$E = \frac{1}{n}\sum_{i=1}^{n}(n - i) = \frac{n - 1}{2}$$

因此，删除算法的时间复杂度为O(n)

顺序表的表示与实现

- **顺序表的操作**
  - 操作3：查找操作——查找指定的元素
    - ✓算法思想：
      - 依次取出顺序表中的元素和指定的元素进行比较，如果相等，则返回其位置；

```
//算法描述
Status  Locate(SqList &L,ElemType e){
    i=1;
    p=L.elem;
    while(i<=L.length&&!(*compare)(*p++,e))
    {
        ++i;
    }
    if(i<=L.length){
      return i;
    }
}//end Locate
```

    - ✓算法分析：
      - 查找成功,比较i次，否则L.length
      - 时间复杂度为:O(L.length)

顺序表的表示与实现

- **顺序表的操作**
  - 操作4：有序表的合并——课后练习
    - ✓分析
    - ✓算法思想
    - ✓算法描述
    - ✓算法分析

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

链表的表示与实现

- **链表：用一组任意的存储单元存储线性表的数据元素（可以连续，也可以不连续）。**
- **结点：数据域与指针域。**
  - 数据域：存放结点本身的数据元素信息；
  - 指针域：存储直接后继存储位置的域。

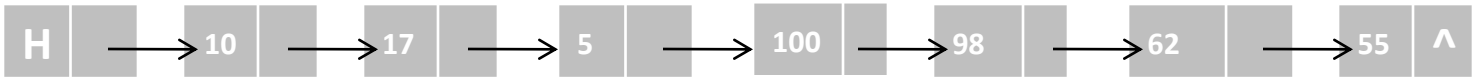
数据域	指针域

13

头指针H

- **示例： (10,17,5,100,98,62,55)**

存储地址	数据域	指针域
1	17	10
10	5	20
13	10	1
20	100	50
25	55	NULL
50	98	55
55	62	25



链表的表示与实现

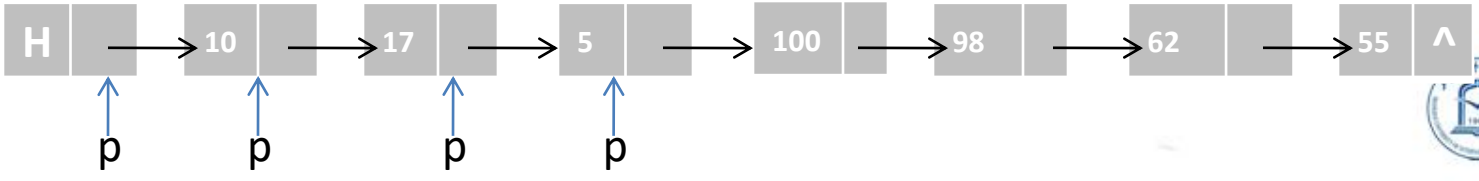
- **链表的存储结构描述**
  - 头结点：在第一个结点前附设一个一个结点，该结点数据域可以不存储信息，也可以存储链表长度等，指针域指向第一个节点。
  - 空表：头结点L的指针域为“NULL”。即：
    - ✓L->next=NULL;


```
//线性表的单链表存储结构
Typedef struct LNode{
    ElemType data;//数据域
    struct Lnode *next;  //指针域
}Lnode, *LinkList;
```

p→data=a_i
p→next指向第i+1个结点,p结点的后继
P→next→data=a_{i+1}

链表的表示与实现

- **链表的操作**
  - 操作1：取元素——取链表中的第i个元素
    - ✓分析：从头指针开始依次往后进行访问，进行计数。当等于i时候返回。
    - ✓算法思想：
      - 初始化指针p指向第一个结点和计数器
        - ◆p=L→next; j=1;
      - 移动指针，进行计数
        - ◆p=p→next;++j;
        - ◆条件：while(p&&j<i)
      - 如果p指向空或者计数器大于i, 查找失败
        - ◆if(!p||j>i) return ERROR;
      - 查找成功，返回
        - ◆e=p->data;



链表的表示与实现

- **链表的操作**
  - 操作1：取元素——取链表中的第i个元素
    - ✓算法描述

```
Status  GetElem(SqList  L,int i,ElemType& e){
    //L为带头结点 的单链表的头结点
    p=L→next;
     j=1;  //j为计数器
     //遍历，沿指针后移
     while(p&&j<i){
          p=p→next;
          ++j;
      }
      //第i个元素不存在
       if(!p||j>i){
          return ERROR;
       }
       //返回第i个元素
       e=p→data;
       return OK;
}
```

算法分析：

表长：n

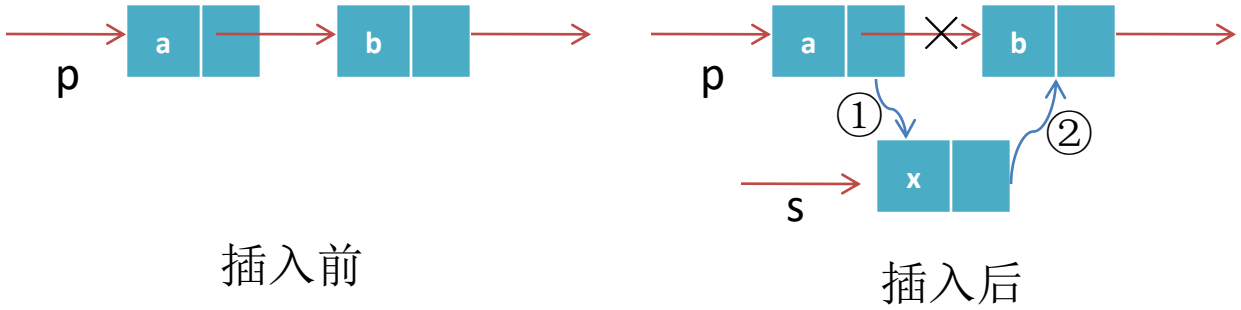
若1<=i<=n，查找成功的次数为i-1;

否则，查找次数为n

时间复杂度为O(n)

链表的表示与实现

- **链表的操作**
  - 操作2：插入操作——第i个元素前插入新的元素
    - ✓分析:



问题1：在第i个结点之前插入，查找时候，查找第i个结点?

——找到第i-1个结点，Why?

问题2：找到结点后（如上图右所示），如何插入？上图右中①②执行顺序如何？怎么执行?

s→next=p→next;

p→next=s;

链表的表示与实现

- **链表的操作**
  - 操作2：插入操作——第i个元素前插入新的元素
    - ✓算法思想:
      - Step 1:依次查找第i-1个元素

        p=L; j=0;

        while(p&&j<i-1){p=p→next;++j}
      - Step 2:查找失败，则返回——插入失败

        if(!p||j>i-1){return ERROR}
      - Step 3:生成新的待插入的结点

        s=(LinkList)malloc(sizeof(LNode));

        s→data=e;
      - Step 4:插入到链表中

        s→next=p→next;

        p→next=s;

链表的表示与实现

- **链表的操作**
  - 操作2：插入操作——第i个元素前插入新的元素
    - ✓算法描述:

```
Status  Insert(SqList&  L,int i,ElemType e){
    //L为带头结点 的单链表的头结点
    p=L;
     j=0;   //j为计数器
     //遍历，沿指针后移,找第i-1个结点
     while(p&&j<i-1){
          p=p→next;
          ++j;
      }
      //i小于1或者大于表长+1
       if(!p||j>i-1){
          return ERROR;
       }
       s=(LinkList)malloc(sizeof(LNode));
       s→data=e;
       //插入s
       s→next=p→next;
       p→next=s;
       return OK;
}
```

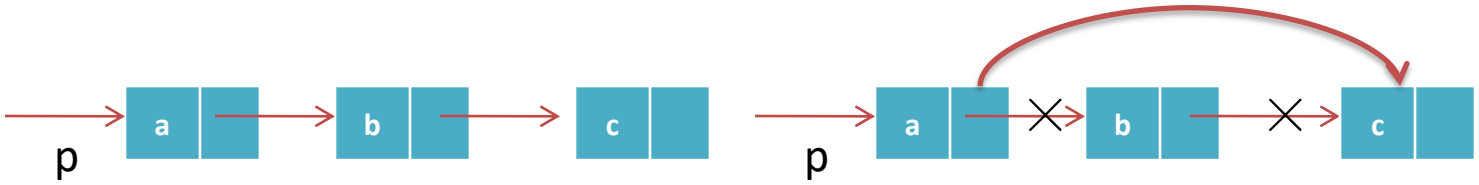
算法分析：

查找第i-1个结点

时间复杂度为O(n)

链表的表示与实现

- **链表的操作**
  - 操作3：删除操作——删除第i个元素
    - ✓分析:



删除前　　删除后

(1)查找第i-1个结点（和插入一样）

(2)如何删除?

只需要改变指针的指向：

p→next=p→next→next; 是否可以? Why?

*****删除前需要保存待删除的结点，否则无法释放。**

```
q=p→next;
p→next=q→next;
free(q);
```

链表的表示与实现

- **链表的操作**
  - 操作3：删除操作——删除第i个元素
    - ✓算法思想:
      - Step 1:依次查找第i-1个元素

        p=L; j=0;

        while(p→next&&j<i-1){p=p→next;++j}
      - Step 2:查找失败，则返回——插入失败

        if(!p→next||j>i-1){return ERROR}
      - Step 3:从链表中删除结点（改变指针方向）

        q=p→next;

        p→next=q→next;
      - Step 4:释放结点

        e=q→data;

        free(q);

链表的表示与实现

- **链表的操作**
  - 操作3：删除操作——删除第i个元素
    - ✓算法描述:

```
Status Delte(SqList&  L,int i,ElemType &e){
    //L为带头结点 的单链表的头结点
    p=L;
     j=0;   //j为计数器
     //遍历，沿指针后移,找第i-1个结点
     while(p→next&&j<i-1){
          p=p→next;
          ++j;
      }
      //删除位置不合理
       if(!p→next||j>i-1){
          return ERROR;
       }
       q=p→next; //保存待删除的结点
       //删除q
       p→next=q→next;
       e=q→data;
       free(q);
       return OK;
}
```

算法分析：

查找第i-1个结点

时间复杂度为O(n)

链表的表示与实现

- **链表的操作**
  - 操作4：建立链表——逆序建立链表
    - ✓分析:
    - ✓从空表开始，依次建立各元素的结点，逐个插入到链表中。
      - 逆序的好处：每次插入的时候都只需要插入在表头；
    - ✓算法思想:
      - Step 1: 建立空表
      - Step 2:从表尾到表头逆序重复建立
      - Step 3:建立新结点，插入到链表中

链表的表示与实现

- **链表的操作**
  - 操作4：建立链表——逆序建立链表
    - ✓算法描述：

```
void Create(LinkList &L,int  n){
    L=(LinkList)malloc(sizeof(LNode));
    L→next=NULL; //建立带头结点的空表
    for(int i=n;i>0;--i){
      //生成新结点
       p=(LinkList)malloc(sizeof(LNode));
       scanf(&p→data);
       p→next=L→next;
       L→next=p;//表头处插入
    }
} //End Create
```

算法分析：

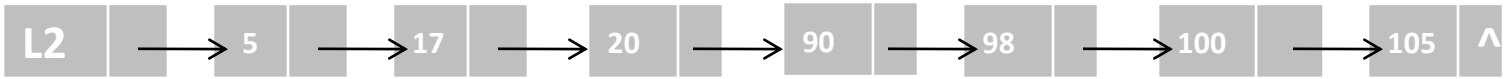
时间复杂度：O(n)

思考题：

如果顺序建立链表，如何实现？

链表的表示与实现

- **链表的操作**
  - 操作5：有序链表的合并——课后练习
    - ✓分析
    - ✓算法思想
    - ✓算法描述
    - ✓算法分析



顺序表与链表的比较

比较内容	顺序表	链表
存储地址	一定连续	连续不连续都可以
存储空间	不需要额外开销	需要额外开销，存指针
取元素操作	下标索引，O(1)	不能下标索引，头指针开始遍历，O(n)
插入操作	需要移动元素，O(n)	不需要移动元素，O(n)
删除操作	需要移动元素，O(n)	不需要移动元素，O(n)

学会对比！

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

静态链表

- 有些程序设计语言没有“指针”。
- 静态链表：利用一维数组来描述链表。
- 数组的一个分量表示结点，用游标（cur）代替指针指向结点在数组中的相对位置。

```
类型说明：
#define MAXSIZE 1000 //链表的最大长度
typedef struct{
    ElemType data;
    int cur;
}componet,SLinkList[MAXSIZE];
```

头结点

序号	元素	游标
0		1
1	a_1	2
2	a_2	3
3	a_3	4
	....	
	a_n	0
	...	

最后一个结点

静态链表

- **S[0]:头结点**
- **S[0].cur指示第一个结点在数组中的位置**
- **若：i=S[0].cur，**
  - 则S[i].data表示第一个数据元素(相当于p→data)
  - S[i].cur指向第二个结点的位置(相当于p→next)
- **i=S[i].cur：相当于p=p→next**

静态链表

- **操作举例——静态链表查找指定元素e的位置**

```
//在静态单链表中查找第1个值为e的元素
//若找到返回它在L的位序，否则返回0
int Locate(SLinkList S,ElemType e){
   //指向静态表中的第一个结点
  i=S[0].cur;
  while(i&&(S[i].data!=e)){
       i=S[i].cur;
   }
    return i;
}
```

算法分析：

时间复杂度：O(n)

- **课后练习：实现静态链表的插入和删除等操作——和单链表类似**

目录

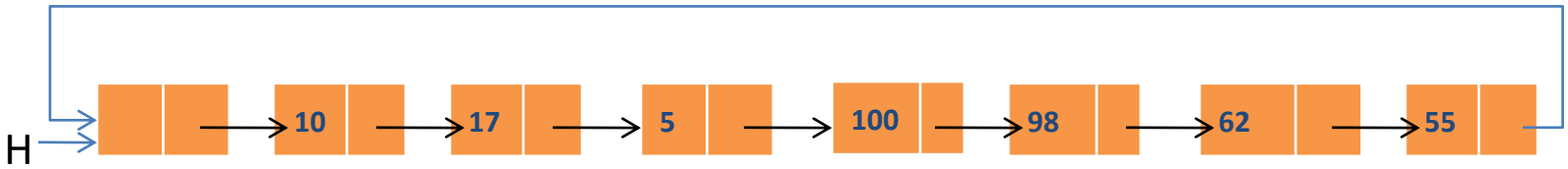
- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

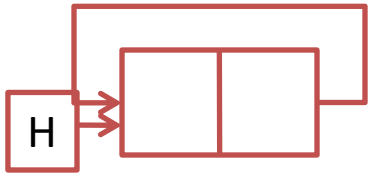
- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

循环链表

- **循环链表：表中的最后一个节点的指针域指向头结点，整个链表形成一个环。**



- **空表：**



  - 空表条件:H→next=H
- **操作：**
  - 操作和线性链表类似，不同点是循环条件不是p或p→next条件为NULL,而是等于头指针。
- **Note:**
  - 循环链表也可以只设立尾指针，可以简便操作，例如：两个线性表的合并，只需要将一个表的尾和另一个表的头相连就可以，运算时间为O(1)。

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

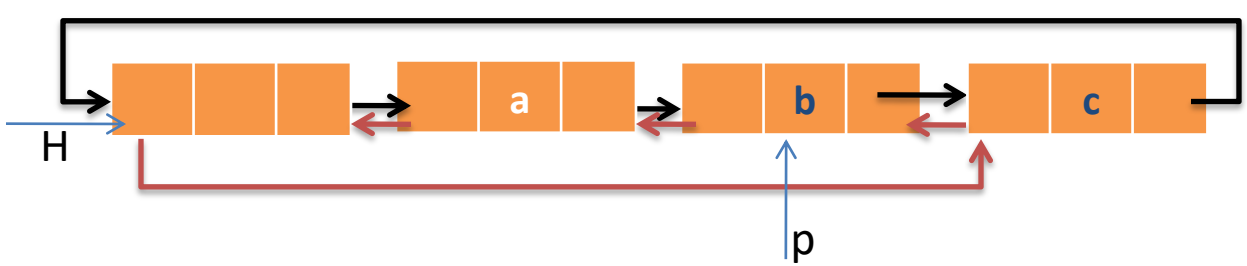
双向链表

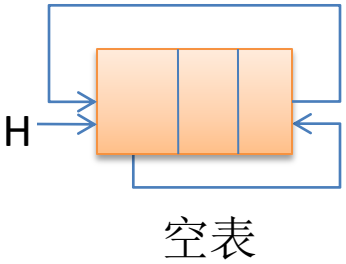
- **单链表：**
  - 查找某个节点后继时间复杂度为O(1)
  - 查找其前驱时间复杂度为O(n)：需要从头指针往后找
- **双向链表：**
  - 设定两个指针域，一个指向其直接后继，一个指向其直接前驱。


```
//双向链表
Typedef struct DNode{
    ElemType data;//数据域
    struct Lnode *prior; //指针域
    struct Lnode *next;  //指针域
}DNode, *DuLinkList;
```

双向链表

- **双向链表查找：**
  - 查找某个节点后继时间复杂度为O(1)
  - 查找其前驱时间复杂度为O(1)
- **双向循环链表**

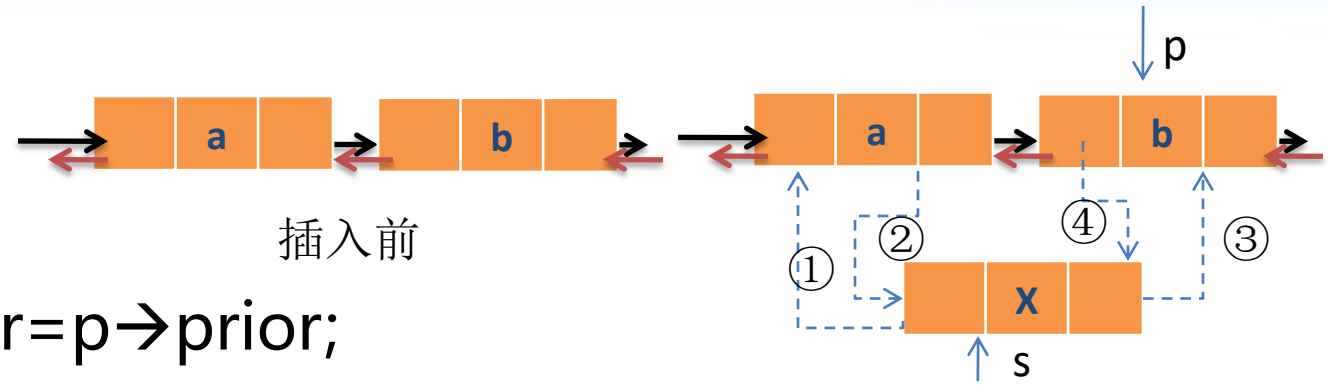




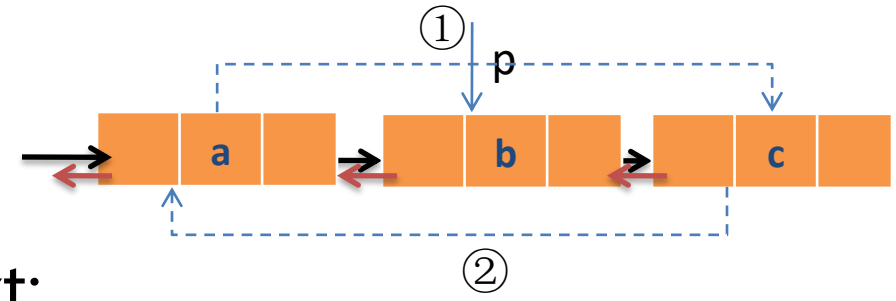
- **空的双向循环链表：H→ next=H or H→prior=H**
- **p→next→prior=p→prior→next=p**

双向链表

- **插入结点：**



  - ①s→prior=p→prior;
  - ②p→prior→next=s;
  - ③s→next=p;
  - ④p→prior=s;
- **删除结点**
- ①p→prior→next=p→next;
- ②p→next→prior=p→prior;



目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

目录

- 线性表的定义
- 顺序表的表示与实现
- 链表的表示与实现
- 静态链表
- 循环链表
- 双向链表
- 应用

应用

- **应用举例：一元多项式的表示与相加**
  - 一元多项式表示：
    - ✓ 例如：$P_n(x)=p_0+p_1x+p_2x^2+\ldots+p_nx^n$
    - ✓ 线性表$P=(p_0,p_1,p_2,\ldots,p_n)$
    - ✓ $Q_n(x)=q_0+q_1x+q_2x^2+\ldots+q_nx^n$
    - ✓ 线性表$Q=(q_0,q_1,q_2,\ldots,q_n)$
  - 一元多项式表示：
    - ✓ $R=P_n(x)+Q_n(x)$
    - ✓ $R=(p_0+q_0,p_1+q_1,p_2+q_2,\ldots,p_n+q_n)$
  - **问题：**
    - ✓ 例如：$P(x)=1+10x^{100}+2x^{50000}$
    - ✓ 用顺序表表示，太多的为0的项，浪费空间！
    - ✓ 解决办法：？？？

指数隐含在系数里

数据域存储系数和指数

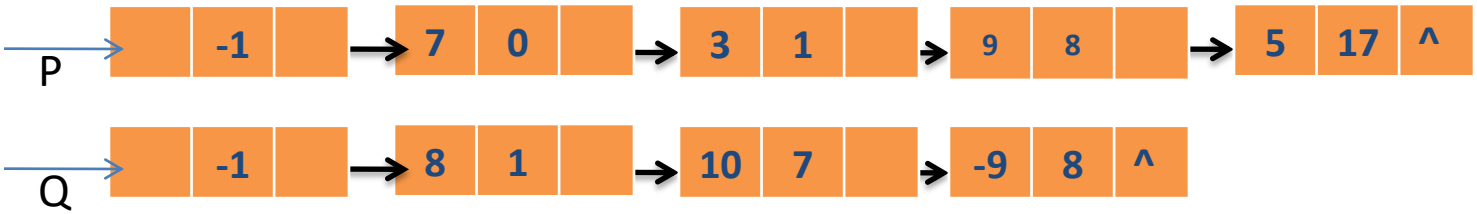
应用

- **应用举例：一元多项式的表示与相加**
  - 表示2：
    - ✓$P=((p_1,e_1),(p_2,e_2),\ldots,(p_m,e_m))$
  - 可以采用顺序存储结构和链式存储结构存放，
    - ✓如果只是求值，不改变系数和指数的运算，用顺序存储结构；
    - ✓否则，采用链式存储结构；
  - 链式存储结构：


```
//一元多项式的存储结构
Typedef struct PNode{
    float coef;//系数
     int expn;//指数
    struct Lnode *next;  //指针域
}PNode, *PolyLinkList;
```

应用

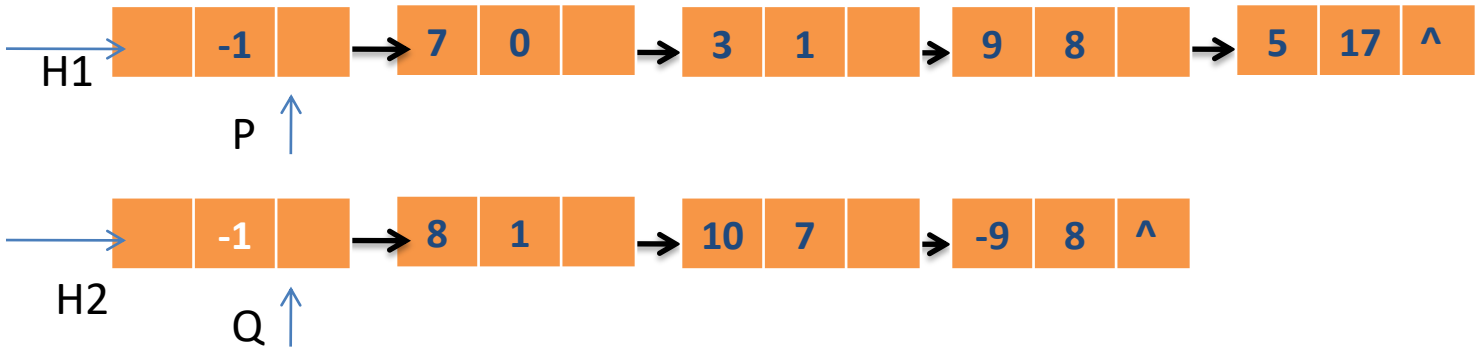
- **应用举例：一元多项式的表示与相加**
  - 操作——求和



  - 算法思想：
  - Step 1:分别从两个链表的头指针P,Q开始，取出当前结点
  - Step 2:比较指数：
    - ✓ (1)如果相等，则系数相加，更新P指向的结点，P，Q后移，同时释放Q指向的结点；
    - ✓ Note:如果和为0，要释放P 指向的结点
    - ✓ (2)如果P.expn<Q.expn，则P后移
    - ✓ (3)如果P.expn>Q.expn，则在P之前插入Q，Q后移。
  - Step 3: 如果一个到达表尾，只需要将未达表尾剩余的插入到链表中。

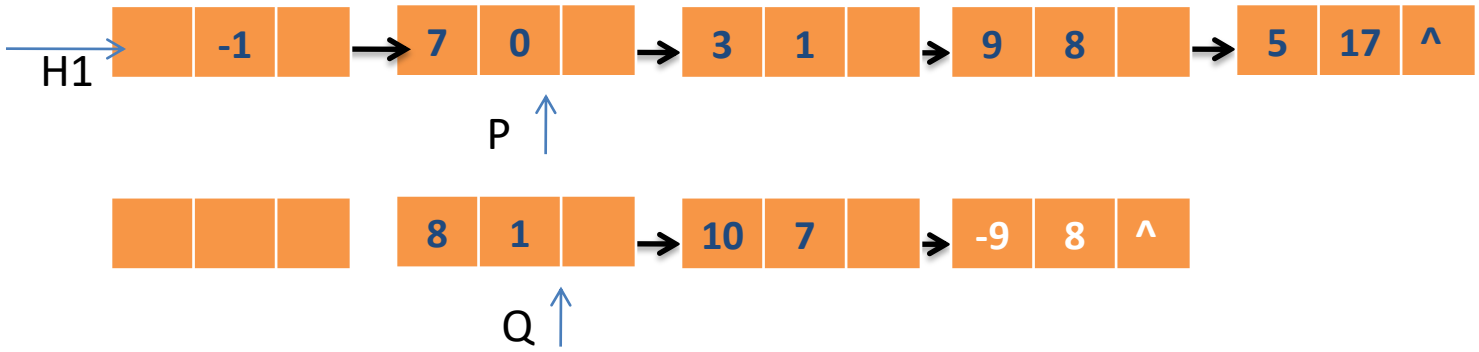
应用

- **应用举例：一元多项式的表示与相加**
  - 执行过程



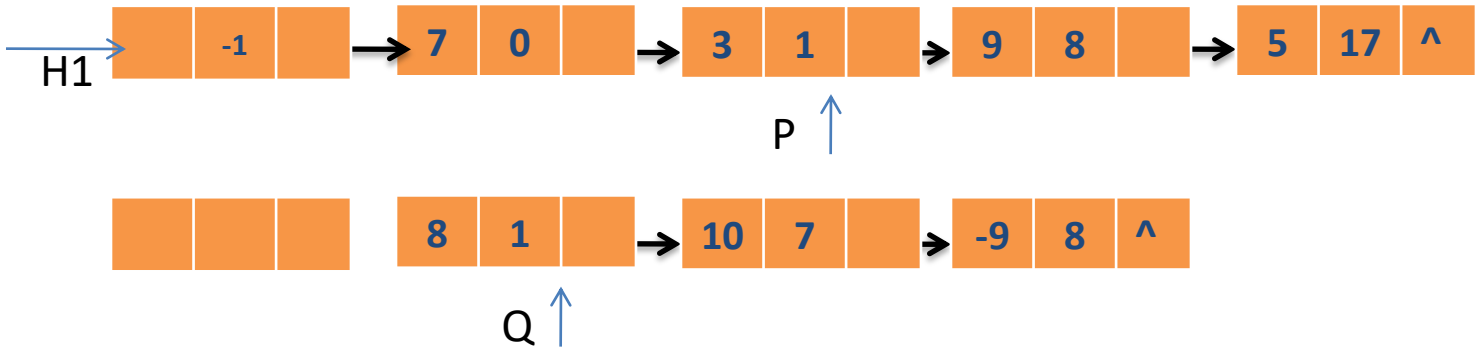
应用

- **应用举例：一元多项式的表示与相加**
  - 执行过程



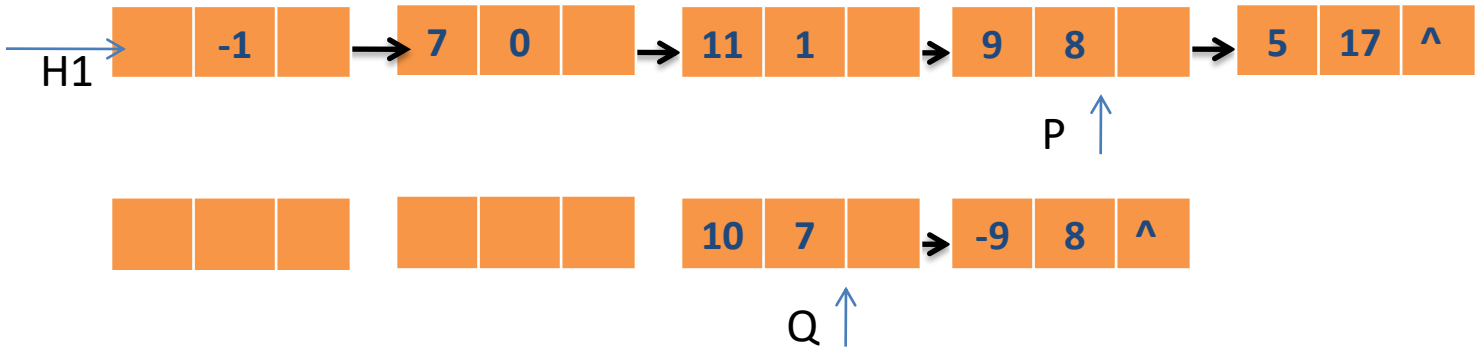
应用

- **应用举例：一元多项式的表示与相加**
  - 执行过程



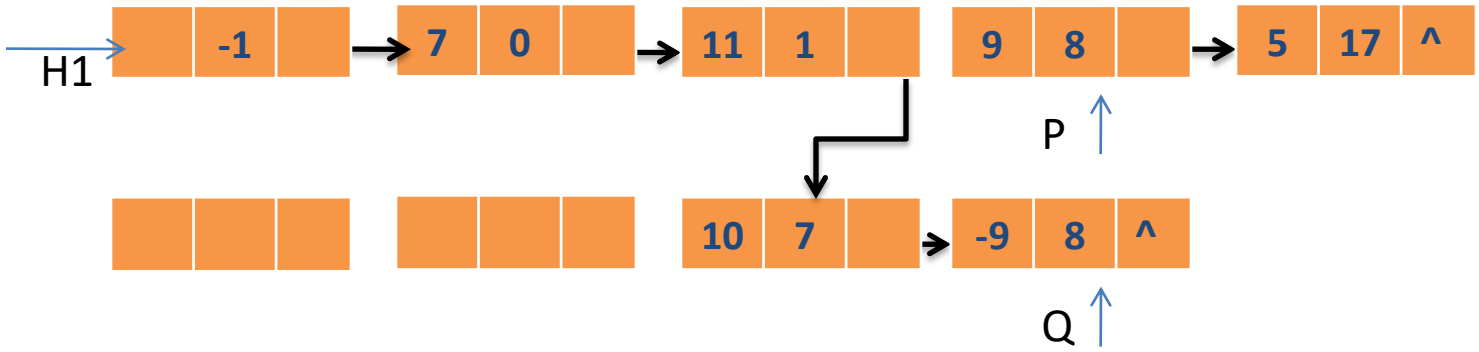
应用

- **应用举例：一元多项式的表示与相加**
  - 执行过程



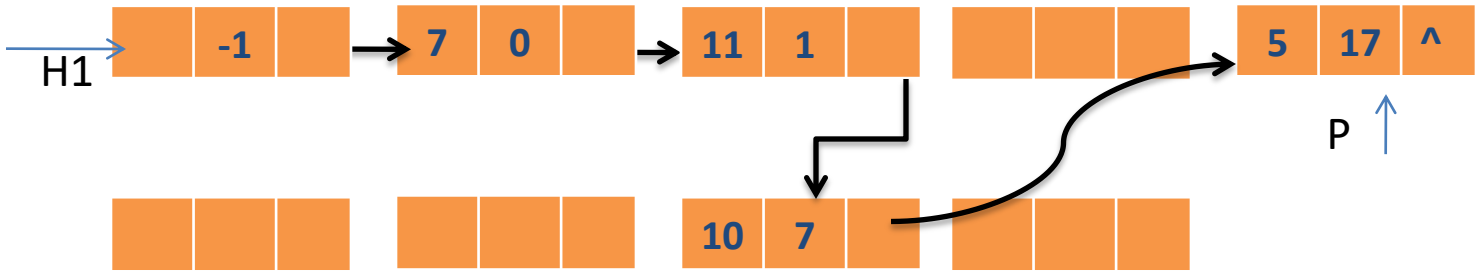
应用

- **应用举例：一元多项式的表示与相加**
  - 执行过程



应用

- **应用举例：一元多项式的表示与相加**
  - 执行过程



应用

- **应用举例：一元多项式的表示与相加**
  - 算法描述
  - 问题：当p→expn >q→expn，需要在p之前插入q结点，此时如何找到p的前驱？
    - ✓通过头指针查找s→next==p?
    - ✓处理办法：利用一个指针s始终指向最新插入的结点。初始化为pl1的头结点！

算法分析：

时间复杂度：O(n1+n2)

```
//两个一元多项式的求和
Status  AddPloy(PolyLinkList& pl1,
PolyLinkList& pl2){
    p=pl1→next;
    q=pl2→next;
     s=pl1;//用s保存为p的前驱
    while(p&&q){
      if(p→expn<q→expn){
        s→next=p;
        s=p;
        p=p→next;
      }else if(p→expn>q→expn){
         s→next=q;
         s=q;
         q=q→next;
      }
      else{ //指数相等，系数求和
           p→coef=p→coef+q→coef;
           if(p→coef==0){
                t1=p;t2=q;
                 p=p→next;
               q=q→next;
               free(t1);
               free(t2);
           }
       }
    }
    if(p){
       s→next=p;
    }
   if(q){
       s→next=q;
    }
}
```

小结

- **线性表的定义与性质：**
  - 一对一的关系
  - 简单常用的结构
- **线性表的顺序存储结构与链式存储结构**
  - 顺序存储结构：地址连续的存储空间存储线性表
  - 链式存储结构：地址不一定连续，通过指针指向
- **顺序表的抽象数据类型定义与操作***
  - 顺序表的插入和删除操作，算法复杂度为O(n)，需要移动元素，取第个元素的操作为O(1)
- **链表的抽象数据类型定义与操作***
  - 链表的插入和删除操作，算法复杂度为O(n)，不需要移动元素，查找第i元素的操作为O(n)
- **循环链表**
- **双向链表**

课后作业

- 利用线性表实现两个一元多项式的乘法