Chapter 2 Linear Structure

Lists, Stacks, Queues, String and Array

学习目标

- 掌握线性表的逻辑结构，线性表的顺序存储结构和链式存储结构的描述方法；熟练掌握线性表在顺序存储结构和链式存储结构的结构特点以及相关的查找、插入、删除等基本操作的实现；并能够从时间和空间复杂性的角度综合比较两种存储结构的不同特点。
- 掌握栈和队列的结构特性和描述方法，熟练掌握栈和队列的基本操作的实现，并且能够利用栈和队列解决实际应用问题。
- 掌握串的结构特性以及串的基本操作，掌握针对字符串进行操作的常用算法和模式匹配算法。
- 掌握多维数组的存储和表示方法，掌握对特殊矩阵进行压缩存储时的下标变换公式，了解稀疏矩阵的压缩存储表示方法及适用范围。
- 了解广义表的概念和特征。

2.1 Lists

- Definition and operations
- Implementation
- Applications

2.1.1 Definition

A list is defined as a sequence of $a_1, \ldots, a_n$,$n(n \geq 0)$.

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

L=('a','b',…,'x','y','z')

L=('good','student','work')

a_i（$1 \leq i \leq n$）称为数据元素；下角标i表示该元素在线性表中的位置或序号。n为线性表中元素的个数，称为表长度。

Student name list

姓　名	学　号	性　别	年龄	成绩
王小林	790631	男	18	88
陈　红	790632	女	20	95
刘建平	790633	男	21	76
张立立	790634	男	17	90
……	……	……	……	……

逻辑特征

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

- ✓ 有限性：个数是有穷的。
- ✓ 相同性：元素类型相同。
- ✓ 相继性：

  a_i为表中第一个元素，无 前驱元素，a_n为表中最后一个元素，无后继元素；

  对于$1<i<n$，a_{i-1}为a_i的直接前驱，a_{i+1}为a_i的直接后继。

Operations

设**L是类型为线性表实例，e 的型为数据元素**实例，**i 为位置变量。**

1）LenList (L);

2）GetElem (L,i);

3）SearchElem (L,e);

4）InsertElem (L,i,e);

5）DeleteElem (L,i)

ADT Sqlist

Data

D={a_i,a_i∈**ElementType,i=1,2,…,n,n≥0}**

R={<a_{i-1},a_i>|a_{i-1},a_i∈D,i=2,…,n}

Operations

求表的长度

LenList(e)

输出：e是整数类型；

后件：返回表的长度。

取表L中第i个数据元素赋值给e

GetElem(i,e)

输入：位置参数i；

输出：e是ElementType类型；

前件：1≤i≤LenList(L)

后件：e被赋予表中第i个位置的元素值。

在L中第i个位置插入新的数据元素e

InsertElem(i,e)

输入：位置参数i, e是ElementType类型；

前件：1≤i≤LenList(L)+1

后件：在L中第i个位置插入新的数据元素e，表长加1。

删除表中第i个数据元素

DeleteElem(i)

输入：位置参数i；

前件：1≤i≤LenList(L)

后件：删除L中第i个位置的数据元素，表长减1。

按值查找

SearchElem(e,i)

输入：e是ElementType的类型；

输出：位置参数i；

前件：1≤LenList(L)

后件：若e存在表中，返回第一个e的位置为i，否则返回0。

End ADT

2.1.2 Physical structure design

1. Sequence -- Storage cell with continuous location address

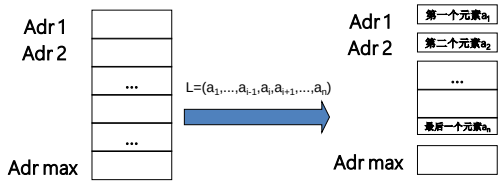
L=(a_1,…,a_{i-1},a_i,a_{i+1},…,a_n)

数据元素的值及其逻辑关系，线性的逻辑关系，可以利用物理位置的相邻表示逻辑上的相邻关系。。

存储方式：将线性表中的元素依次存放在连续的存储空间。

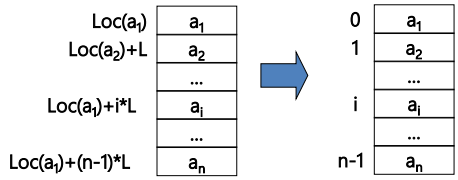
1. Sequence -- Storage cell with continuous location address

存储方式：将线性表中的元素依次存放在连续的存储空间。



1. Sequence --Array implementation

$LOC(a_i) = LOC(a_1) + (i-1) * L \quad 1<=i<=n$



13

Characters:

How to express the logical structure in the computer?

元素之间逻辑上的相继关系，用物理上的相邻关系来表示（用物理上的连续性刻画逻辑上的相继性）。

逻辑上相邻的元素，在物理位置上也相邻。

是一种随机访问存取的结构，也就是说可以按元素的位置之间进行访问，其位置可以由公示直观的计算出来。

14

Implementation：利用数组结构实现

```
Template <class List_entry>
Class list {
  public:
    List();
    int size() const;
    bool full() const;
    bool empty() const;
    ……

  protected:
    int last;
    List_entry entry[max_list];
}
```

15

Operations

16

Insert element *e* at location *i*

InsertElem(*i*,*e*)

Input： *i* is location, *e* is ElementType；

Pre-condition： $1 \le i \le L.last+1$

Post-condition： Insert element *e* at location *i* and add 1 to Length of *L*

17

- 插入前：$(a_1, \dots, a_{p-1}, a_p, \dots, a_n)$
- 插入后：$(a_1, \dots, a_{p-1}, x, a_p, \dots, a_n)$

a_{p-1}和a_p之间的逻辑关系发生了变化

⇩

顺序存储要求存储位置反映逻辑关系

⇩

存储位置要反映这个变化

18

```
int InsertElem(sqlist *L,DataType x,int i)
{ int j;
 if (((*L).last)>=maxsize－1)
   { printf(“overflow\n”); return -1;} \\溢出
 else
  if((i<1)||(i>((*L).last)＋1)
    { printf(“error\n”); return -1;}  \\非法位置
  else
   {   for (j=(*L).last;j>=i;j--)
         (*L).data[j+1]＝(*L).data[j];
       (*L).data[i]＝x;
       (*L).last＝(*L).last+1;
     }
  return(1);
}
```


Delete element at location i

DeleteElem(i)
Input：location i;
Pre-condition：1≤i≤L.last
Post-condition：delete the element at location i.


```
int  DeleteElem(sqlist *L,int i)
{ int j;
  if ((i<1)||(i>(*L).last+1))
    { printf(“error\n”); return -1;}  \\非法位置
  else
   {   for(j=i;j<=(*L).last;j++)
         (*L).data[j-1]＝(*L).data[j];
       (*L).last--;              \\表长减1
   }
  return(1);
}
```


Analysis

```
int InsertElem(sqlist *L,DataType x,int i)
{ int j;
 if (((*L).last)>=maxsize－1)
   { printf(“overflow\n”); return -1;} \\溢出
 else
  if((i<1)||(i>((*L).last)＋1)
    { printf(“error\n”); return -1;}  \\非法位置
  else
   {   for (j=(*L).last;j>=i;j--)
         (*L).data[j+1]＝(*L).data[j];
       (*L).data[i]＝x;
       (*L).last＝(*L).last+1;
     }
  return(1);
}
```



```
int  DeleteElem(sqlist *L,int i)
{ int j;
  if ((i<1)||(i>(*L).last+1))
    { printf(“error\n”); return -1;}  \\非法位置
  else
   {   for(j=i;j<=(*L).last;j++)
         (*L).data[j-1]＝(*L).data[j];
       (*L).last--;              \\表长减1
   }
  return(1);
}
```


Analysis of operations(Insert and delete)

- 与表长及位置有关。
- 插入：
  - 最坏：i=1，移动次数为n
  - 最好: i=表长+1，移动次数为0
  - 平均：等概率情况下，平均移动次数n/2
- 删除：
  - 最坏：i=1，移动次数为n-1
  - 最好: i=表长，移动次数为0
  - 平均：等概率情况下，平均移动次数(n-1)/2

Search(Value)

LocateElem1(*e*,*i*)
Input: element *e*;
Output: location of element *e*;
Pre-condition: *L* is not empty;
Post-condition: If *e* is in List L
then return location *i*.


```
int LocateElem1(SqList L, DataType e)
{ i=1;
  while ( i<=L.last && e != L.data[i-1]) ++i;
  if (i<=L.last) return i-1;
       else return -1;
}
```


Search(location)

LocateElem1(*i*,*e*)
Input: location *i*;
Output: element *e*;
Pre-condition: 1≤*i*≤L.last
Post-condition: return the element
at location *i*.

Search(2)

```
DataType LocateElem2(SqList L, int i)
{
 If ((i<1)||(i>((*L).last))
   { printf("error\n");
    return -1;}  \\非法位置
  return (*L).data[i-1];
 }
```


Analysis of implementation in array

1）优点
- 顺序表的结构简单
- 顺序表的存储效率高，是紧凑结构,无须为表示节点间的逻辑关系而增加额外的存储空间
- 顺序表是一个随机存储结构（直接存取结构）

2）缺点
- 在顺序表中进行插入和删除操作时，需要移动数据元素，算法效率较低。
- 对长度变化较大的线性表，或者要预先分配较大空间或者要经常扩充线性表，给操作带来不方便。

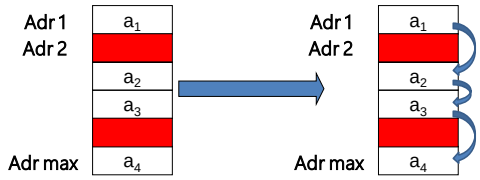
逻辑上相邻的元素，在物理位置上也相邻。
（进行插入删除操作的原因）

2. Linked

In the computer

散列在计算机中的存储单元，地址不一定连续。存储单元通过保存存储的地址来关联。

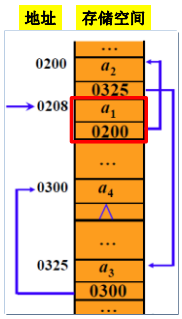
2. Linked $L=(a_1,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n)$



散列在计算机中的存储单元，地址不一定连续。存储单元通过保存存储的地址（指针）来关联。

一个存贮单元由结点组成，每个结点含有存放元素的数据域和存放其逻辑关系的指针域，这样就形成链接式存储结构。

(a1, a2 ,a3, a4)的存储示意图

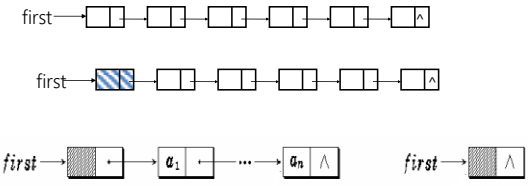


Linked : Logical relationship is described by point

后继（或前驱）信息：需要通过指针的概念进行表示。

One Point: | Data | Point |

Express of linked list

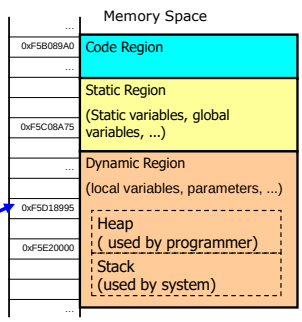


Characters of linked list:

3. Implementation: point type

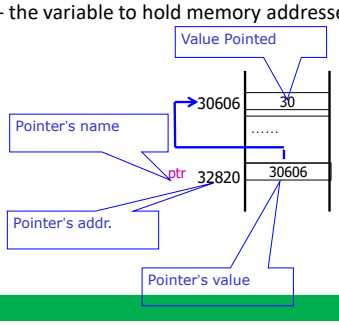
- Memory space
  - A "continuous" space
  - Three regions
- Memory address
  - Each memory unit has an address
- Variable address
  - The initial address of the memory allocated for the variable

int i=0;



About a Pointer

- *Pointer:*
  - the variable to hold memory addresses as its value



Declare a Pointer

- Pointer is a composite data type
- Syntax:

dataType * pointerName;

Pointer type | Pointer symbol "*" | Pointer name

- For example, a pointer named pCount that can point to an int value:

  int *pCount;

指针的基本操作

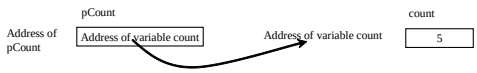
"&"：取地址运算符
"*"：取值运算符
在定义指针时，"*"是指针说明符，表示指针
在使用指针时，"*"表示去该指针所指向变量的值

- int count=5;

  int *pCount = &count;

  int *ptr = NULL;

Address-of operator "&". The value of "&count" is the address of count.




```
int a;
int *p = &a;    // p指向a
*p = 10;        // 相当于 a = 10;
```

```
int a, *p;
p = &a;
*p = 10;
a++;
printf ("a = %d, *p = %d", a, *p);
```

输出结果：
a = 11, *p = 11

例如：

int n=2,*pointer;

pointer=&n; (*pointer=?)


```
#include <stdio.h>
void main( )
{
  int  a=3,b=5;
  int *p= &a;
  printf("%d\n",*p);
  *p=4;
   p=&b;
  printf("%d\n",*p);
  *p=6;
  printf("a=%d,b=%d\n",a,b);
}
```

输出结果为：
3
5
a=4, b=6

Examples

```
void main()
{
    int a =10, b = 20;
    int *pa, *pb;
    pa = &a; //Assign the address directly
    pb = pa+2;

    cout <<"a in: "<< &a <<endl;
    cout <<"b in: "<< &b <<endl;

    cout <<"pa is: "<< pa <<endl;
    cout <<"pa points to: "<< *pa <<endl;
    cout <<"pb is: "<< pb <<endl;
    cout <<"pb points to: "<< *pb <<endl;
    if(pa != pb)
        cout <<"pa and pb are not equal!"<<endl;
    cout<<"pb - pa is "<<pb-pa<<endl;
}
```

```
a in: 0012FF34
b in: 0012FF2C
pa is: 0012FF34
pa points to: 10
pb is: 0012FF3C
pb points to: 4227998
pa and pb are not equal!
pb - pa is 2
```


Cautions

- The type of the address assigned to a pointer must be consistent with the pointer type.

  For example,

  int area = 1;

  double *pArea = &area;

- The white space besides "*" is optional

  int*ptr; int *ptr; int* ptr; int * ptr;

- One "*" for one pointer

  int *ptr1, *ptr2; //two pointers

  int* ptr1, ptr2; // one pointer, one integer

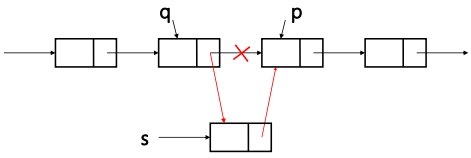
Implementation: point type

```
Template <class List_entry>
Struct Node {
    Node_entry entry;
    Node<Node_entry> * next;
    Node();
    Node(Node_entry,Node<Node_entry> *link=NULL);
}
```



```
Template <class List_entry>
Class list {
 public:
   List();
   int size() const;
   bool full() const;
   bool empty() const;
   ……

 protected:
   int count;
   Node<List_entry>  *head;
}
```


Insert
（在p结点所在的位置插入/在p结点前插入）

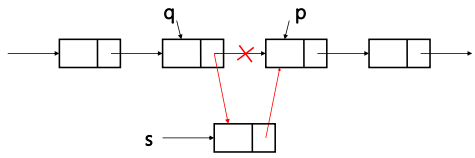


建立新结点s，向新结点中添入内容

查找插入结点的位置p,以及p结点的前驱结点q

将新结点链入链中

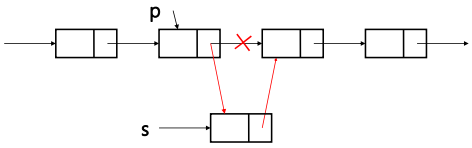
Insert（在p结点所在的位置插入/在p结点前插入）




```
New(S);
S->data=a;
S->next=p;
q->next=S;
```

```
New(S);
S->data=a;
q->next=S;
S->next=p;
```


Insert（在p结点之后插入）




```
New(S);
S->data=a;
p->next=S;
S->next=p->next;
```

```
New(S);
S->data=a;
S->next=p->next;
p->next=S;
```

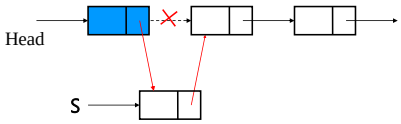

Operations of Linked list

- Create
- Search
- Delete
- Union

Create

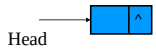
Algorithm:

1. **To create Head**
2. **To create new node S**
3. **Insert S**



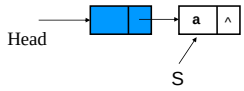
Case: L=(a,b,c,d,e)

Step 1



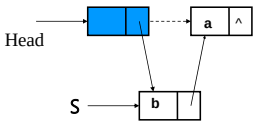
New(Head);
Head->next=NULL;

Step 2: insert a

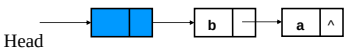


New(S);
S->data=a;
S->next=Head->next;
Head->next=S;

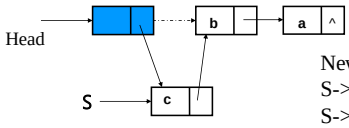
Insert b



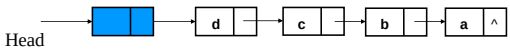
New(S);
S->data=b;
S->next=Head->next
Head->next=S;



Insert c



New(S);
S->data=b;
S->next=Head->next;
Head->next=S;



Algorithm

1. **To create Head**
2. **To create S**
3. **Insert S**

```
Void CreateList(Head)
{
  new(Head);
  Head->next=NULL;
  scanf("%c",ch);
  while (ch<>'#') do
    {
      new(S);
      S->data=ch;
      S->next=Head->next;
      Head->next=S;
    }
}
```



```
Void CreateList(Head)
{ new(Head);
  Head->next=NULL;
  Last=Head;
  scanf("%c",ch);
  while (ch<>'#') do
    {  new(S);
       S->data=ch;
       S->next=NULL;
       Last->next=S;
       (              )=S;
    }
}
```

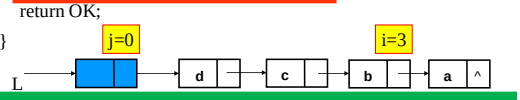
自查：熟练单链表上的操作

建立单链表
单链表上按位置查找
单链表上按值查找
删除单链表中的元素
两个递增有序的单链表合并

Insert element at location i：

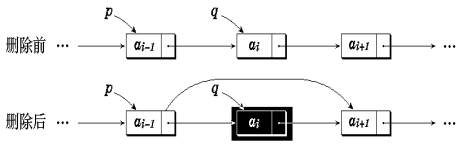
```
Status ListInsert_L(LinkList &L,int i,DataType e)
{  LinkList p,s;
   p=L;  int j=0;
   while (p && j<i-1) { p=p->next; j=j+1;}
   if (!p) return ERROR;
   s = (LinkList) malloc( sizeof (LNode));
   s->data = e; s->next = p->next;
   p->next = s;
   return OK;
}
```

问题：带头结点？



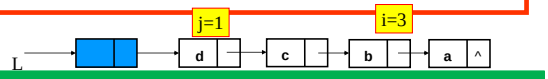
Delete

```
Status ListDelete_L(LinkList &L, int i, DataType &e)
{  LinkList p,q;
   p=L;  int j=0;
   while (p->next && j<i-1) { p=p->next; ++j;}
   if (!(p->next)) return ERROR;
   q=p->next;  p->next = q->next;
   e=q->data;  free(q);
   return OK;
}
```



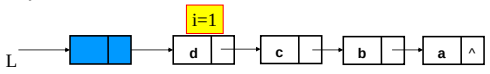
Search (i)

```
Status GetElem_L(LinkList L, int i,DataType &e)
{  LinkList p;
   p=L->next;  int j=1;
   while (p && j<i) { p=p->next; ++j; }
   if (!p) return ERROR;
   e=p->data;
   return OK;
}
```

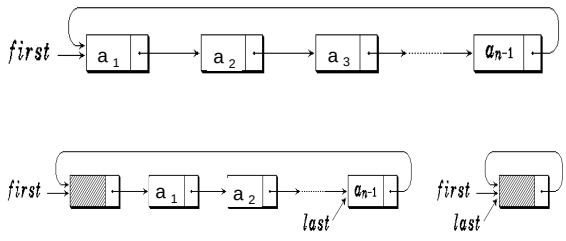


Search (e)

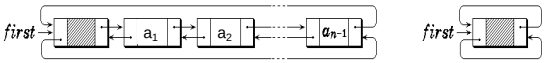
```
int LinkLocate_L (LinkList L, int x)
{ int i;   LinkList  p;
  p=L->next;  i=1;
  while (p!=NULL && p->data != x)
        {p= p->next; i++;}
  if (!p)  { printf ("Not found! \n");
             return(0);
           }
  else { printf ("i=%d\n",i);return (i); }
}
```



Circular linked list



Doubly Linked list



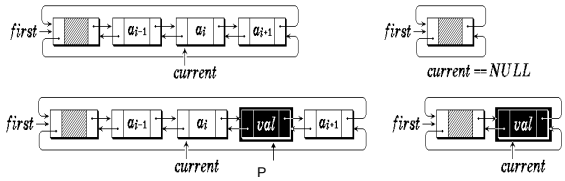
p == p→prior→next == p→next→prior

p→prior = p ; p→next = p;

61

Insert at doubly circular Linked list

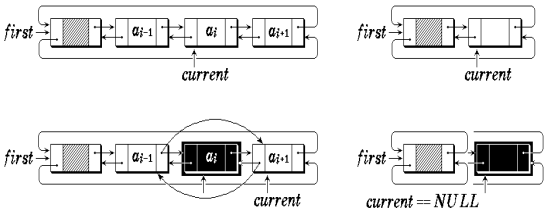
p→prior = current; (1)
p→next =current→next; (2)
current→next = p; (3)
p→next→prior = p; (4)



62

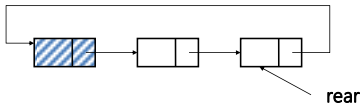
Delete

current→next→prior = current→prior;
current→prior→next = current→next;

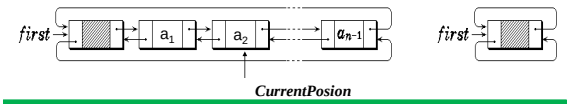


63

With point rear



With current location



64

连续设计和链接设计的对比

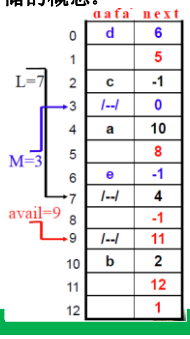
比较参数	连续设计	链接设计
表的容量	固定，不易扩充	灵活，易扩充
存取操作	随机访问存取	顺序访问存取
时间	插入删除费时间	访问元素费时间
空间	估算长度，浪费空间	实际长度

65

4. Implement Linked list in array

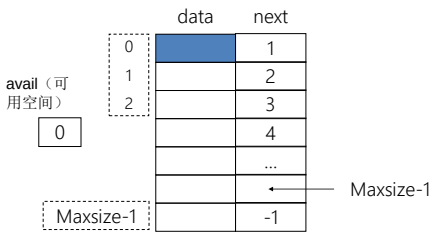
利用静态数组实现链接存储的概念。

举例：
线性表 L=(a,b,c)
线性表 M=(d,e)
空闲表 avail=9



66

结点说明



67

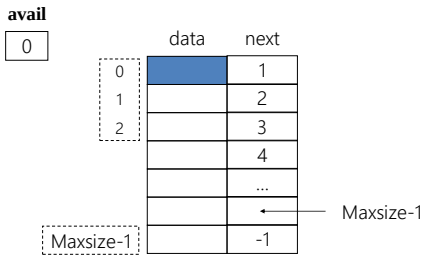
```
#define  maxsize  1024    \\存储池最大容量
typedef int datatype;
typedef int cursor;
typedef struct            \\结点类型
{ datatype data;
  cursor next;
} node;

node nodepool[maxsize];    \\存储池
cursor avail;
```

68

Initial array nodepool

```
INITIALIZE()
{ int i;
  for (i=0;i<maxsize-1;i++)
    nodepool[i].next＝i+1;
  nodepool[maxsize-1].next＝-1;
  avail ＝0;
}
```

69



70

```
结点的分配算法
cursor GETNODE()
{ cursor p;
  if (avail ==-1) p＝-1;
  else
    { p＝ avail;
      avail ＝nodepool[avail].next;
    }
  return p;
}
```

71

```
结点的回收算法
FREENODE(cursor p)
{
  nodepool[p].next＝ avail;
  avail ＝p;
}
```

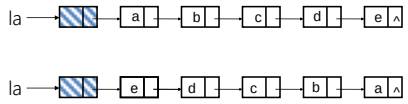
72

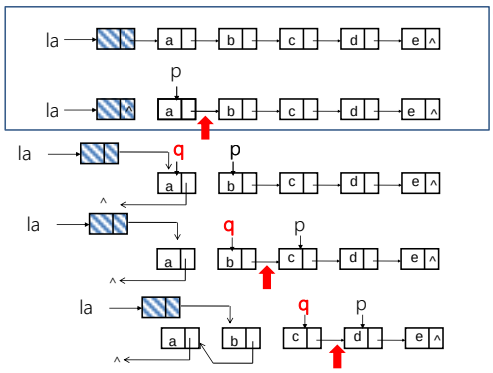
静态链表查找算法

```
cursor FINDSTLIST(cursor L,int i)
{ cursor p; int j;
  p=L; j=0;
  while ((nodepool[p].next!=-1)&&(j<i))
   { p=nodepool[p].next;
     j++;
   }
  if (i==j) return(p);
  else return -1;
}
```


2.1.3 Applications

Case 1. 以单链表存储结构实现线性表的就地逆转






```
Void convers (linklist *la)
{ linklist *p,*q;
  If (la->next !=NULL) {
    p = la->next;
    la->next = NULL;
   While (p != NULL);
   {  q = p;
      p = p->next;
      q->next = la->next;
      la->next = q;
}
}
}
```


Case 2. 一元多项式的表示和相加

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

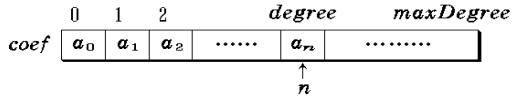
- *n*阶多项式$P_n(x)$有*n*+1项。
  - 系数 $a_0, a_1, a_2, \ldots, a_n$
  - 指数 0, 1, 2, …, *n*。按升幂排列

1. 第一种表示方法

$P_n=(a_0, a_1, a_2, \ldots, a_n)$

适用于指数连续排列、“0”系数较少的情况。

但对于指数不全的多项式，如$P_{20000}(x) = 3 + 5x^{50} + 14x^{20000}$, 会造成系统空间的巨大浪费。



2. 第二种表示方法

一般情况下，一元多项式可写成：

$P_n(x)=p_1x^{e1}+p_2x^{e2}+...+p_mx^{em}$

其中：p_i是指数为ei的项的非零系数，

0≤e1 ≤e2 ≤... ≤em ≤n

二元组表示 $((p_1,e1),(p_2,e2), ... ,(p_m,em))$

例：$P_{999}(x) = 7x^3 - 2x^{12} -8x^{999}$

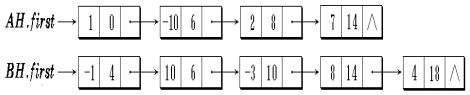
表示成: ((7,3),(-2,12),(-8,999))

	0	1	2		i		m
$coef$	a_0	a_1	a_2	……	a_i	……	a_m
exp	e_0	e_1	e_2	……	e_i	……	e_m

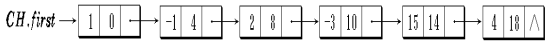
3.第三种表示方法--多项式链表

$AH = 1 - 10x^6 + 2x^8 +7x^{14}$

$BH = - x^4 + 10x^6 - 3x^{10} + 8x^{14} +4x^{18}$



(a) 两个相加的多项式



(b) 相加结果的多项式

算法思想：

初始化；
While (两个链都没处理完)
{ if (指针指向当前节点的指数项相同)
{系数相加，在C链中填加新的节点；
A、B链的指针均前移；}
else
{以指数小的项的系数添入C链中的新节点；
指数小的相应链指针前移；}
}
While(A链处理完)
{ 顺序处理B链；}
While(B链处理完)
{ 顺序处理A链；}

小 结

- 表的逻辑结构及其特点
- 表的物理结构及其特点
- 表的基本操作算法及性能分析
- 应用实例