

数据结构与算法	第三章 线性表	3—1
<h2>第三章 线性表 (Liner List)</h2>		
3.1 抽象数据型线性表		
3.2 线性表的实现		
3.3 栈 (Stack)		
3.4 队列 (Queue)		
3.5 串 (String)		
3.6 数组 (Array)		
3.7 广义表 (Generalized List)		

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—2
知识点：		
■ 线性表的逻辑结构和各种存储表示方法 ■ 定义在逻辑结构上的各种基本运算（操作） ■ 在各种存储结构上如何实现这些基本运算 ■ 各种基本运算的时间复杂性		

重点：

- 熟练掌握顺序表和单链表上实现的各种算法及相关的时间复杂性分析

难点：

- 使用本章所学的基本知识设计有效算法解决与线性表相关的问题

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—3
<h3>3.1 抽象数据型线性表</h3>		
【定义】	线性表是由 $n(n \geq 0)$ 个相同类型的元素组成的有序集合。	

记为：

$$(a_1, a_2, a_3, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n)$$

其中：① n 为线性表中元素个数，称为线性表的长度；

当 $n=0$ 时，为空表，记为 $()$ 。

② a_i 为线性表中的元素，类型定义为elementtype

③ a_1 为表中第1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 $\dots, a_{i-1}, a_i, a_{i+1} \dots (1 \leq i < n)$ ，称 a_{i-1} 为 a_i 的直接前驱， a_{i+1} 为 a_i 的直接后继。（位置概念！）

④ 线性表是有限的，也是有序的。

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—4
<h3>3.1 抽象数据型线性表</h3>		
线性表 $LIST = (D, R)$	数学模型	

$$D = \{ a_i \mid a_i \in Elementset, i = 1, 2, \dots, n, n \geq 0 \}$$

$$R = \{ H \}$$

$$H = \{ \langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i = 2, \dots, n \}$$

操作：设 L 的型为 $LIST$ 线性表实例， x 的型为 $elementtype$ 的元素实例， p 为位置变量。所有操作描述为：

- ① **INSERT(x, p, L)**
- ② **LOCATE(x, L)**
- ③ **RETRIEVE(p, L)**
- ④ **DELETE(p, L)**
- ⑤ **PREVIOUS(p, L), NEXT(p, L)**
- ⑥ **MAKENULL(L)**
- ⑦ **FIRST(L)**

哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—5
<h3>3.1 抽象数据型线性表</h3>		
<p>举例：设计函数 DELEVAL (LIST &L, elementtype d)，其功能为删除 L 中所有值为 d 的元素。</p> <pre>void DELEVAL(List &L, elementtype d) { position p; p = FIRST(L); while (p != END(L)) { if (same(RETRIEVE(p, L), d)) DELETE(p, L); else p = NEXT(p, L); } }</pre>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—6
<h3>3.2 线性表的实现</h3>		
<p>问题：确定数据结构（存储结构）实现型LIST，并在此基础上实现各个基本操作。</p> <p>存储结构的三种方式：</p> <ul style="list-style-type: none"> ① 连续的存储空间（数组） → 静态存储 ② 非连续存储空间——指针（链表） → 动态存储 ③ 游标（连续存储空间+动态管理思想） → 静态链表 		
<h4>3.2.1 指针和游标</h4> <p>指针：地址量，其值为另一存储空间的地址；</p> <p>游标：整型指示量，其值为数组的下标，用以表示指定元素的地址；或位置；或位置的下标（所在的数组下标）</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—7
<h4>3.2.2 线性表的数组实现</h4>		
<p>顺序表： 把线性表的元素逻辑顺序依次存放在数组的连续单元内，再用一个整型量表示最后一个元素所在单元的下标。</p> <p>特点：</p> <ul style="list-style-type: none"> ■ 元素之间的逻辑关系（相继/相邻关系）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）； ■ 是一种随机存储结构，也就是可以随机存取表中的任意元素，其存储位置可由一个简单直观的公式来表示。 <p>图3-1 线性表的数组实现</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—8
<h4>3.2.2 线性表的数组实现</h4>		
<p>类型定义：</p> <pre>#define maxlen 100 struct List { elementtype elements [maxlen]; int last; };</pre> <p>位置类型：</p> <pre>typedef int position;</pre> <p>空单元线性表 L：</p> <pre>List L;</pre> <p>表示：</p> <pre>L.elements[p] // L的第p个元素 L.last L的长度，最后元素的位置</pre>		
<p>图3-1 线性表的数组实现</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—9
3.2.2 线性表的数组实现		
操作：		
<pre> ① void INSERT (elementtype x, position p, LIST &L) { position q; if (L.last >= maxlen - 1) error(表满); else if ((p > L.last + 1) (p < 1)) error(指定位置不存在); else { for (q = L.last; q >= p; q--) L.elements[q + 1] = L.elements[q]; L.last = L.last + 1; L.elements[p] = x; } } //时间复杂性: O(n) </pre>		
图 线性表的数组实现 哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—10
3.2.2 线性表的数组实现		
<ol style="list-style-type: none"> ② position LOCATE (elementtype x , LIST L) <pre> { position q; for (q = 1; q <= L.last; q++) if (L.elements[q] == x) return (q); return (L.last + 1); } //时间复杂性: O(n) </pre> 		
<ol style="list-style-type: none"> ③ elementtype RETRIEVE (position p , LIST L) <pre> { if (p > L.last) error(指定元素不存在); else return (L.elements[p]); } //时间复杂性: O(1) </pre> 		
图 线性表的数组实现 哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—11
3.2.2 线性表的数组实现		
<ol style="list-style-type: none"> ④ void DELETE(position p , LIST &L) <pre> { position q; if ((p > L.last) (p < 1)) error(指定位置不存在); else { L.last = L.last - 1; for (q = p; q <= L.last; q++) L.elements[q] = L.elements[q + 1]; } } //时间复杂性: O(n) </pre> 		
<ol style="list-style-type: none"> ⑤ position PREVIOUS(position p , LIST L) <pre> 空单元 if ((p <= 1) (p > L.last)) error(前驱元素不存在); else return (p - 1); } //时间复杂性: O(1) </pre> 		
图 线性表的数组实现 哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—12
3.2.2 线性表的数组实现		
<ol style="list-style-type: none"> position NEXT(position p , LIST L) <pre> { if ((p < 1) (p >= L.last)) error(前驱元素不存在); else return (p + 1); } //时间复杂性: O(1) </pre> 		
<ol style="list-style-type: none"> ⑥ position MAKENULL(LIST &L) <pre> { L.last = 0; return (L.last + 1); } //时间复杂性: O(1) </pre> 		
<ol style="list-style-type: none"> ⑦ position FIRST(LIST L) <pre> { return (1); } //复杂性: O(1) </pre> 		
<ol style="list-style-type: none"> ⑧ position END(LIST L) <pre> { return (L.last + 1); } // O(1) </pre> 		
图3-1 线性表的数组实现 哈尔滨工业大学计算机科学与技术学院 张岩		

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法 第三章 线性表 3—13

3.2.3 线性表的指针实现

单链表：一个线性表由若干个结点组成，每个结点均含有两个域：
存放元素的信息域和存放其后继结点的指针域，这样就形成一个
单向链接式存储结构，简称单向链表或单向线性链表。

结构特点：

- 逻辑次序和物理次序不一定相
同；
- 元素之间的逻辑关系用指针表
示；
- 需要额外空间存储元素之间
的关系；
- 非随机存储结构

```
LIST header;
position p, q;
```

结点形式

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—14

3.2.3 线性表的指针实现

类型定义：

```
struct celltype {
    elementtype element;
    celltype *next;
}; /* 结点型 */
/* 线性表的型 */
typedef celltype *LIST;
/* 位置型 */
typedef celltype *position;
```

结点形式

记法： $a_2: (*p).element$; $a_2: p \rightarrow element$;
 $q: (*p).next$; $q: p \rightarrow next$;

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—15

3.2.3 线性表的指针实现

操作讨论： 插入元素：

```
q = new celltype;
q->element = x;
q->next = p->next;
p->next = q;
```

或：

```
temp = p->next;
p->next = new celltype;
p->next->element = x;
p->next->next = temp;
```

(a) 表头插入元素

(b) 中间插入元素

(c) 表尾插入元素

讨论表头结点的作用

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—16

3.2.3 线性表的指针实现

操作讨论： 删除元素：

```
q = p->next;
p->next = q->next;
delete q;
```

(a) 删除第一个元素

或：

```
q = p->next;
p->next = p->next->next;
delete q;
```

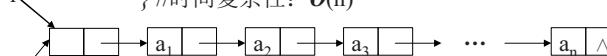
(b) 删除中间元素

(c) 删除表尾元素

讨论结点 a_i 的位置 p

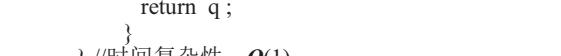
哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—17
3.2.3 线性表的指针实现		
操作： ① position END (LIST) { position q; q = L; while (q->next != NULL) q = q->next; return (q); } //时间复杂性: $O(n)$		
 ② void INSERT (elementtype x, position p, LIST &L) { position q; q = new celltype; q->element = x; q->next = p->next; p->next = q; } //时间复杂性: $O(1)$		

数据结构与算法	第三章 线性表	3—18
3.2.3 线性表的指针实现		
操作： ③ position LOCATE (elementtype x, LIST L) { position p; p = L; while (p->next != NULL) if (p->next->element == x) return p; else p = p->next; } //时间复杂性: $O(n)$		
 ④ elementtype RETRIEVE (position p, LIST L) { return (p->next->element); } //时间复杂性: $O(1)$		

数据结构与算法	第三章 线性表	3—19
3.2.3 线性表的指针实现		
操作： ⑤ void DELETE (position p, LIST &L) { position q; if (p->next != NULL) { q = p->next; p->next = q->next; delete q; } } //时间复杂性: $O(1)$		
 position PREVIOUS (position p, LIST L) { position q; if (p == L->next) error ('不存在前驱元素'); else { q = p->next; while (q->next != p) q = q->next; return q; } } //时间复杂性: $O(n)$		

数据结构与算法	第三章 线性表	3—20
3.2.3 线性表的指针实现		
操作： ⑥ position NEXT (position p, LIST L) { position q; if (p->next == NULL) error ('不存在后继元素'); else { q = p->next; return q; } } //时间复杂性: $O(1)$		
 ⑦ position MAKENULL (LIST &L) { L = new celltype; L->next = NULL; return L; } //时间复杂性: $O(1)$		

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法 第三章 线性表 3—21

3.2.3 线性表的指针实现

操作: ⑧ position FIRST (LIST L)
 {
 return L;
 } //时间复杂性: O(1)

举例: 遍历线性链表, 按照线性表中元素的顺序, 依次访问表中的每一个元素, 每个元素只能被访问一次。

顺序存储 固定, 不易扩充 随机存取 插入删除费时间 估算表长度, 浪费空间	比较参数 ←表的容量→ ←存取操作→ ←时间→ ←空间→	链式存储 灵活, 易扩充 顺序存取 访问元素费时间 实际长度, 节省空间
--	--	--

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—22

3.2.4 线性表的游标实现

静态链表: 把线性表的元素存放在数组的单元中 (不一定按逻辑顺序连续存放), 每个单元不仅存放元素本身, 而且还要存放其后继元素所在的数组单元的下标 (游标)。

L [7] → M [3] → 9 → available

结点形式: spacestr
 element | next
 ↓↓
 结点信息 下一结点位置

0	d	6
1		5
2	c	-1
3	--	0
4	a	10
5		8
6	e	-1
7	--	4
8		-1
9	--	11
10	b	2
11		12
12		1

多个线性表共用一个存储池 存储池

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—23

3.2.4 线性表的游标实现

类型定义:

```
typedef struct {
    elementtype element;
    int next;
} spacestr; /*结点类型*/
spacestr SPACE[ maxsize ]; /*存储池*/ M [3]
typedef int position, cursor;
cursor av; /*游标变量, 标识线性表*/

```

线性表: L = (a, b, c) L = 7 空闲表: available = 9

元素: M[i].element → i型; i为elementtype(基本、复合)
 下一元素位置: SPACE[i].next → i型; i为int
 类似指针链表, 对游标实现的线性表仍设表头结点。

存储池

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—24

3.2.4 线性表的游标实现

SPACE

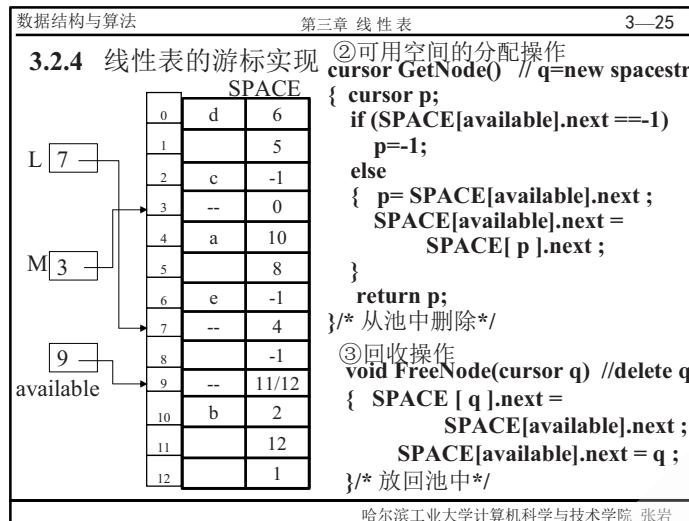
0	---	1
1		2
2		3
3		4
4		5
5		6
6		7
7		8
8		9
9		10
10		11
11		12
12		-1

存储池的管理 ①可用空间的初始化

```
void Initialize()
{
    int j;
    /* 依次链接池中结点 */
    for (j=0; j<maxsize-1; j++)
        SPACE[j].next=j+1;
    /* 最后一个接点指针域为空 */
    SPACE[j].next=-1;
    /* 标识线性表 */
    available=0;
}
```

哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126



数据结构与算法 第三章 线性表 3—26

3.2.4 线性表的游标实现

操作:

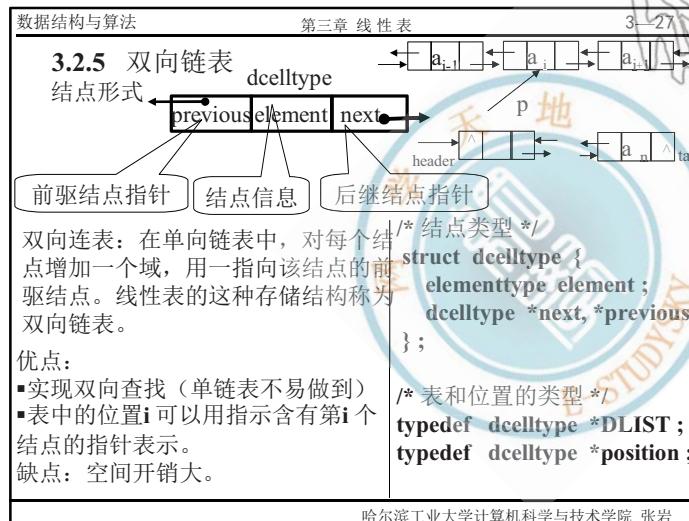
```

④ void INSERT( elementtype x, position p, spacestr *SPACE )
{
    position q;
    q = GetNode();
    SPACE[ q ].element = x;
    SPACE[ q ].next = SPACE[ p ].next;
    SPACE[ p ].next = q;
    p->next = q;
}

⑤ void DELETE ( position p, spacestr *SPACE )
{
    position q;
    if ( SPACE[ p ].next != -1 )
        p->next != NULL
    {
        q = SPACE[ p ].next;
        SPACE[ p ].next = SPACE[ q ].next;
        p->next = q->next;
        FreeNode( q );
    }
}

```

哈尔滨工业大学计算机科学与技术学院 张岩



数据结构与算法 第三章 线性表 3—28

3.2.5 双向链表

操作:

```

void XINSERT( elementtype x, position p, DLIST &L )
{
    position q;
    q = new dcelltype;
    q->element = x;
    p->previous->next = q;
    q->previous = p->previous;
    q->next = p;
    p->previous = q;
}

```

删除位置p的元素:

```

void DELETE( position p, DLIST &L )
{
    if (p->previous!=NULL)
        p->previous->next = p->next;
    if (p->next!=NULL)
        p->next->previous = p->previous;
    delete p;
}

```

哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法 第三章 线性表 3—29

3.2.6 环形链表

对线性链表的改进，解决“单向操作”的问题；改进后的链表，能够从任意位置元素开始，访问表中的每一个元素。

单向环形链表：在(不带表头结点)的单向链表中，使末尾结点的指针域指向头结点，得到一个环形结构；用指向末尾结点的指针标识这个表。

存储结构：与单向链表相同(略)

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—30

3.2.6 环形链表

操作：①在表左端插入结点 $\text{INSERT}(x, \text{FIRST}(R), R) \rightarrow \text{LINSERT}(x, R)$

```
void LINSERT(elementtype x, LIST &R)
{
    celltype *p;
    p = new celltype;
    p->element = x;
    if (R == NULL)
    {
        p->next = p; R = p;
    }
    else
    {
        p->next = R->next; R->next = p;
    }
}
```

②在表右端插入结点 $\text{INSERT}(x, \text{END}(R), R) \rightarrow \text{RINSERT}(x, R)$

```
void RINSERT(elementtype x, LIST R)
{
    LINSERT(x, R);
    R = R->next;
}
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—31

3.2.6 环形链表

操作：③从表左端删除结点 $\text{DELETE}(\text{FIRST}(R), R) \rightarrow \text{LDELETE}(R)$

```
void LDELETE(LIST &R)
{
    celltype *p;
    if (R == NULL)
        error ("空表");
    else
    {
        p = R->next;
        R->next = p->next;
        if (p == R)
            R=NULL;
        delete p;
    }
}
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—32

3.2.6 环形链表

双向环形链表：

双向环形链表的结构与双向连表结构相同，只是将表头元素的空previous域指向表尾，同时将表尾的空next域指向表头结点，从而形成向前和向后的两个环形链表，对链表的操作变得更加灵活。

举例：设计算法，将一个单向环形链表反向。头元素变成尾元素，尾元素变成新的头元素，依次类推。

哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法 第三章 线性表 3—33

3.2.6 环形链表

算法如下：存储结构定义略

```
void REVERS( LIST &R )
{ position p, q;
  q = R;
  p= R->next ;
  R= R->next ;
}
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—34

3.3 栈 (Stack)

栈是线性表的一种特殊形式，是一种限定性数据结构，也就是在对线性表的操作加以限制后，形成的一种新的数据结构。

定义：是限定只在表尾进行插入和删除操作的线性表。
栈又称为后进先出 (Last In First Out) 的线性表。
简称LIFO结构。

栈举例

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—35

2.3 栈

栈的基本① MAKENULL (S)
操作 ② TOP (S)
③ POP (S)
④ PUSH (x, S)
⑤ EMPTY (S)

举例：利用栈实现行编辑处理。
设定符号 '#' 为擦讫符，用以删除 '#' 前的字符；符号 '@' 为删除符，用以删除当前编辑行。
原理：
读字符 { 一般字符进栈；
擦讫符退栈；
删除符则清栈 }

算法：

```
void LINEEDIT()
{ STACK S;
  char c;
  MAKENULL (S);
  c = getchar();
  while (c != '\n')
  { if (c == '#')
    POP (S);
    else if (c == '@')
    MAKENULL (S);
    else
    PUSH (c, S);
    c = getchar();
  }
 逆序输出栈中所有元素;
}
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—36

3.3 栈

3.3.1 栈的实现

1、顺序存储
类型定义：

```
enum Boolean{TRUE,FALSE}
typedef struct {
  elementtype elements[maxlength];
  int top;
} STACK;
```

STACK S;

栈的容量： maxlength - 1 ;
栈空： S.top = 0 ;
栈满： S.top = maxlength - 1 ;
栈顶元素： S.elements[S.top] ;

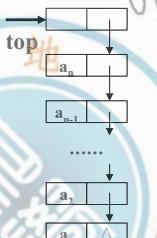
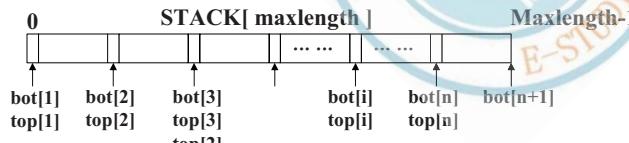
顺序栈示意图

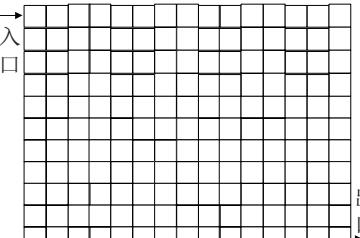
哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—37
3.3.1 栈的实现		
1、顺序存储		
<pre>① void MAKENULL(STACK &S) 操作: { S.top = 0 ; }</pre>		
<pre>② Boolean EMPTY(STACK S) { if (S.top < 1) return TRUE else return FALSE ; }</pre>		
<pre>③ elementtype TOP(STACK S) { if EMPTY(S) return NULL; else return (S.elements[S.top]); }</pre>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—38
3.3.1 栈的实现		
1、顺序存储		
<pre>操作: if (EMPTY (S)) error ("栈空"); else S.top = S.top - 1 ;</pre>		
<pre>④ elementtype POP(STACK &S) { if (EMPTY (S)) error ("栈空"); else S.top = S.top - 1 ;</pre>		
<pre>⑤ void PUSH(elementtype x, STACK &S) { if (S.top == maxlen - 1) error ("栈满"); else { S.top = S.top + 1 ; S.elements[S.top] = x ; }</pre>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—39
3.3.1 栈的实现		
2、链式存储(见第1章)		
<p>采用由指针形成的线性链表来实现栈的存储,要考虑链表的哪一端实现元素的插入和删除比较方便。</p> <p>实现的方式如右图所示,其操作与线性链表的表头插入和删除元素相同。</p>		
3、多个栈共用一个存储空间的讨论		
 		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—40
3.3.2 栈的应用		
1、栈和递归过程P.74 - 75		
2、迷宫求解		
<p>问题:</p> <pre>(010001100011111, 100011011100111 011000011100111 110111101101100 110100101111111 0011011101010111 011110011111111 001101101111101 110001101100000 001111000111110 010011110111110)</pre>		
		
迷宫示例		
<p>一个迷宫可用上图所示方阵[m,n]表示, 0表示能通过, 1表示不能通过。现假设耗子从左上角[1,1]进入迷宫, 编写算法, 寻求一条从右下角 [m,n] 出去的路径。</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法 第三章 线性表 3—41

2、迷宫求解

分析：

(1) 迷宫可用二维数组 $Maze[i,j]$ ($1 \leq i \leq m, 1 \leq j \leq n$) 表示，入口 $Maze[1,1] = 0$ ；耗子在任意位置可用 (i,j) 坐标表示；

(2) 位置 (i,j) 周围有8个方向可以走通，分别记为：E, SE, S, SW, W, NW, N, NE；如图所示。方向 v 按从正东开始且顺时针分别记为1-8, $v=1,8$ ；设二维数组 move 记下八个方位的增量；

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—42

2、迷宫求解

V	i	j	说明
1	0	1	E
2	1	1	SE
3	1	0	S
4	1	-1	SW
5	0	-1	W
6	-1	-1	NW
7	-1	0	N
8	-1	1	NE

从 (i, j) 到 (g, h) 且 $v=2$ (东南) 则
有：
 $g = i + move[2, 1] = i + 1;$
 $h = j + move[2, 2] = j + 1;$

迷宫示例

(3) 为避免时时监测边界状态，可把二维数组 $Maze[1:m, 1:n]$ 扩大为 $Maze[0:m+1, 0:n+1]$ ，且另0行和0列、 $m+1$ 行和 $n+1$ 列的值为1；

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—43

2、迷宫求解

(4) 采用试探的方法，当到达某个位置且周围八個方向走不通时需要回退到上一个位置，并换一个方向继续试探；为解决回退问题，需设一个栈，当到达一个新位置时将 (v, i, j) 进栈，回退时退栈。

(5) 每次换方向寻找新位置时，需测试该位置以前是否已经经过，对已到达的位置，不能重复试探，为此设矩阵 $mark$ ，其初值为0，一旦到达位置 (i, j) 时，置 $mark[i, j] = 1$ ；

文字描述算法：

- 耗子在 $(1, 1)$ 进入迷宫，并向正东 ($v=1$) 方向试探。
- 监测下一方位 (g, h) 。若 $(g, h) = (m, n)$ 且 $Maze[m, n] = 0$ ，则耗子到达出口，输出走过的路径；程序结束。
- 若 $(g, h) \neq (m, n)$ ，但 (g, h) 方位能走通且第一次经过，则记下这一步，并从 (g, h) 出发，再向东试探下一步。否则仍在 (i, j) 方位换一个方向试探。
- 若 (i, j) 方位周围8个方位阻塞或已经过，则需退一步，并换一个方位试探。若 $(i, j) = (1, 1)$ 则到达入口，说明迷宫走不通。

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—44

2、迷宫求解

```

void GETMAZE ( maze , mark ,move ,s )
{ (i, j, v)=(1,1,1); mark[1, 1] = 1; top = 0 ;
  do { g = move[v, 1]; h = move[v, 2];
        if ((g == m) && (h == n) && (maze[m, n] == 0))
          { output(S); return ; }
        if ((maze[g, h] == 0) && mark[g, h] == 0)
          { mark[g, h] = 1; PUSH(i, j, v, s); (i, j, v)=(g, h, 1) ; }
        else if (v < 8)
          v = v + 1;
        else { while ((s.v == 8) && (!EMPTY(s))) POP(s);
                  if (top > 0)
                    (i, j, v++)=POP(s); } ;
      } while ((top) && (v != 8));
      cout << "路径不存在!" << endl;
    }
  
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—45

3、表达式求值

表达式： $\left\{ \begin{array}{l} \text{前缀表达式（波兰式）} \\ \text{中缀表达式} \\ \text{后缀表达式（逆波兰式）} \end{array} \right.$

例如：

$$(a + b) * (a - b) \left\{ \begin{array}{l} *+ab-ab \\ (a+b)*(a-b) \\ ab+ab-* \end{array} \right.$$

高级语言中，采用类似自然语言的中缀表达式，但计算机对中缀表达式的处理是很困难的，而对后缀或前缀表达式则显得非常简单。

后缀表达式的特点：

- ① 在后缀表达式中，变量（操作数）出现的顺序与中缀表达式顺序相同。
- ② 后缀表达式中不需要括弧定义计算顺序，而由运算（操作符）的位置来确定运算顺序。

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—46

3、表达式求值

I. 将中缀表达式转换成后缀表达式

对中缀表达式从左至右依次扫描，由于操作数的顺序保持不变，当遇到操作数时直接输出；为调整运算顺序，设立一个栈用以保存操作符，扫描到操作符时，将操作符压入栈中，进栈的原则是保持栈顶操作符的优先级要高于栈中其他操作符的优先级，否则，将栈顶操作符依次退栈并输出，直到满足要求为止。
遇到‘(’进栈，当遇到‘)’时，退栈输出直到‘)’为止。

II. 由后缀表达式计算表达式的值

对后缀表达式从左至右依次扫描，与 I 相反，遇到操作数时，将操作数进栈保留；当遇到操作符时，从栈中退出两个操作数并作相应运算，将计算结果进栈保留；直到表达式结束，栈中唯一元素即为表达式的值。

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—47

3.4 排队或队列 (Queue)

队列是对线性表的插入和删除操作加以限定的另一种限定性数据结构。

[定义] 将线性表的插入和删除操作分别限制在表的两端进行，和栈相反，队列是一种先进先出（First In First Out，简称 FIFO 结构）的线性表。

队列示意图：

操作：MAKENULL(Q)、FRONT(Q)、ENQUEUE(x, Q)、DEQUEUE(Q)、EMPTY(Q)

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法 第三章 线性表 3—48

3.4 队列 (Queue)

3.4.1 队列的指针实现

元素的 i 型：

```
struct celltype { elementtype element; celltype *next; };
```

队列的 i 型：

```
struct QUEUE { celltype *front; celltype *rear; };
```

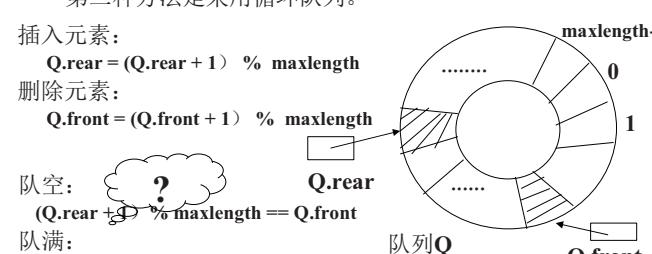
队列的指针实现示意图：

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—49
3.4.1 队列的指针实现		
操作：		
<pre> ① void MAKENULL(QUEUE &Q) { Q.front = new celltype; Q.front->next = NULL; Q.rear = Q.front; } ② Boolean EMPTY(QUEUE Q) ③ elementtype FRONT(QUEUE Q) QUEUE &Q; QUEUE Q; { if (Q.front == Q.rear) { return TRUE; return Q.front->element; else } return FALSE; } } </pre>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—50
3.4.1 队列的指针实现		
<pre> ④ void ENQUEUE (elementtype x, QUEUE &Q) { Q.rear->next = new celltype; Q.rear = Q.rear->next; Q.rear->element = x; Q.rear->next = NULL; } ⑤ void DELETE (QUEUE &Q) { celltype *temp; if (!EMPTY(Q)) error ("空队列"); else { temp = Q.front->next; Q.front->next = temp->next; delete temp; if (Q.front->next == NULL) Q.rear = Q.front; } } </pre>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—51
3.4.2 队列的数组实现		
队列的 <i>i</i> 型实现		
<pre> struct QUEUE { elementtype element [maxlen]; int front; int rear; }; 右图：队列Q状态1 Q [a1 a2 a3 a4 ... an ...] maxlen ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ front rear </pre> <p>下图：队列Q状态2 maxlen 假溢出</p> <p>随着不断有元素出队和进队（插入和删除），队列的状态由1变成2；此时 a_n 占据队列的最后一个位置；第 $n+1$ 个元素无法进队，但实际上，前面部分位置空闲。</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—52
3.4.2 队列的数组实现		
解决假溢出的方法有多种；一是通过不断移动元素位置，每当有元素出队列时，后面的元素前移一个位置，使队头元素始终占据队列的第一个位置。 第二种方法是采用循环队列。		
<p>插入元素： $Q.rear = (Q.rear + 1) \% maxlen$</p> <p>删除元素： $Q.front = (Q.front + 1) \% maxlen$</p> <p>队空： $(Q.rear + 1) \% maxlen == Q.front$</p> <p>队满： $(Q.rear + 1) \% maxlen == Q.front$</p>		
		

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—53
3.4.2 队列的数组实现		
问题：如何解决循环队列中队空与队满状态相同？		
方法一：约定队头指针在队尾指针的下一位置上； 方法二：另设一个标志位用以区别队空与队满两种状态； 结论：两种方法的代价是相同的。		
操作： <pre> int addone(int i) { return ((i + 1) % maxlen); } ① void MAKENULL (QUEUE &Q) { Q.front = 0 ; Q.rear = maxlen - 1; } ② boolean EMPTY(Q) QUEUE Q ; { if (addone(Q.rear) == Q.front) return TRUE ; else return FALSE ; }</pre>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—54
3.4.2 队列的数组实现		
操作：③ elementtype FRONT(QUEUE Q) <pre> { if (EMPTY(Q)) return NULL ; else return (Q.elements[Q.front]); }</pre>		
④ void ENQUEUE (elementtype x, QUEUE Q) { if (addone (addone(Q.rear))==Q.front) error ("队列满"); else { Q.rear = addone (Q.rear); Q.elements[Q.rear] = x ; }}		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—55
3.4.2 队列的数组实现		
⑤ void DEQUEUE (QUEUE Q); { if (EMPTY(Q)) error ("空队列"); else Q.front = addone (Q.front); };		
3.5 多项式的代数运算		
结点结构 <pre> struct polynode { int coef ; int exp ; polynode *link ; }; typedef polynode *polypointer ;</pre>		
系数 指数 下一项地址		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—56
3.5 多项式的代数运算		
例如： 多项式 $p(x) = 3x^{14} + 2x^8 + 1$ 采用链表表示如下：		
<pre> P → [] → [3 14] → [2 8] → [1 0] → [] </pre>		
算法attach(c, e, d) 建立一个新结点，其系数coef=c，指数exp=e；并把它链到d所指结点之后，返回该结点指针。		
<pre> polypointer attach (int c , int e , polypointer d) { polypointer x ; x = new polynode ; x->coef = c ; x->exp = e ; d->link = x ; return x ; };</pre>		
算法padd 实现两个多项式 a, b 相加； $c(x) = a(x) + b(x)$		
哈尔滨工业大学计算机科学与技术学院 张岩		

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—57
3.5 多项式的代数运算		
<pre>polypointer padd() polypointer a, polypointer b; { polypointer p, q, d, c; int x; p = a->link; q = b->link; c = new polynode; d = c; while ((p != NULL) && (q != NULL)) switch (compare (p->exp, q->exp)) { case '<': x = p->coef + q->coef; if (x) d = attach(x, p->exp, d); p = p->link; q = q->link; break; case '>': d = attach(p->coef, p->exp, d); } } q = q->link; break; case '=': d = attach(q->coef, q->exp, d); p = p->link; break; } while (p != NULL) { d = attach(p->coef, p->exp, d); p = p->link; } while (q != NULL) { d = attach(q->coef, q->exp, d); q = q->link; } d->link = NULL; p = c; c = c->link; delete p; return c;</pre>		

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—58
3.5 多项式的代数运算		

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—59
3.6 串 (String)		
3.6.1 抽象数据型串		

串是线性表的一种特殊形式，表中每个元素的类型为字符型，是一个有限的字符序列。

串的基本形式可表示成： $S = 'a_1a_2a_3 \dots a_n'$ ；

其中：char a_i ； $0 \leq i \leq n$ ； $n \geq 0$ ；当 $n = 0$ 时，为空串。

n 为串的长度；

C 语言中串有两种实现方法：

1、字符数组，如：char str1[10]；

2、字符指针，如：char *str2；

操作：

string NULL();

Boolean ISNULL(S);

void IN(S, a);

int LEN(S);

void CONCAT(S1, S2);

string SUBSTR(S, m, n);

Boolean INDEX(S, S1);

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—60
3.6 串 (String)		

例一：将串T 插在串S 中第 i 个字符之后INSERT(S, T, i)。

```
void INSERT( STRING &S, STRING T, int i )
{ STRING t1, t2 ;
  if ((i < 0) || (i > LEN(S)))
    error '指定位置不对';
  else
    if (ISNULL(S)) S = T ;
    else
      if (ISNULL(T))
        { t1 = SUBSTR(S, 1, i);
          t2 = SUBSTR(S, i + 1, LEN(S));
          S = CONCAT(t1, CONCAT(T, t2));
        }
  }
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法

第三章 线性表

3—61

3.6 串(String)

例二：从串 S 中将子串 T 删除 **DELETE(S, T)**。

```
void DELETE( STRING &S, STRING T )
{ STRING t1, t2 ;
  int m, n ;
  m = INDEX( S, T );
  if ( m==0 )
    error ‘串S中不包含子串T’ ;
  else
    { n = LEN( T ) ;
      t1 = SUBSTR( S, 1, m - 1 ) ;
      t2 = SUBSTR( S, m + n, LEN( S ) );
      S = CONCAT( t1, t2 );
    }
}
```

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法

第三章 线性表

3—62

3.6 串 (String)

3.6.2 串的实现

1、串的顺序存储

采用连续的存储空间（数组），自第一个元素开始，依次存储字符串中的每一个字符。

`char str[10] = "China";`

0	1	2	3	4	5	6	7	8	9
'C'	'h'	'i'	'n'	'a'	'.'				

str

串的顺序存储

操作：NULL, ISNULL, IN, LEN, CONCAT, SUBSTR, INDEX

2、串的链式存储

构造线性链表，`element`类型为`char`，自第一个元素开始，依次存储字符串中的每一个字符。

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法

第三章 线性表

3—63

3.6.2 串的实现

2、串的链式存储

```
struct node {
    char data ;
    node *link ;
};

typedef node *STRING1 ;
STRING str1 ;
```

```
struct node {
    char data[4] ;
    node *link ;
};

typedef node *STRING2 ;
STRING str2 ;
```

str1 → [‘C’] → [‘h’] → [‘i’] → [‘n’] → [‘a’]

str2 → [‘C’ ‘h’ ‘i’ ‘n’] → [‘C’ ‘e’ ‘c’ ‘t’]

假设地址量 (**link**) 占用 2 个字节

5/15

5/12

数据结构与算法

第三章 线性表

3—64

2、串的链式存储

操作：

INDEX(S,S1)

若S1是S的子串则返回

S1首字符在S中的位置；

否则，返回0；

```

int INDEX( STRING1 S, S1 )
{
    struct node *p, *q, *i;
    int t;
    if (( S1 != NULL ) && ( S != NULL ))
    {
        t = 1; i = S; q = S1;
        do {
            if ( p->data == q->data )
            {
                q = q->link;
                if ( q == NULL ) return(t);
                p = p->link;
            }
            else
            {
                t = t + 1; i = i->link;
                p = i; q = S1;
            }
        } while ( p != NULL );
    }
    return 0;
}

```

哈尔滨工业大学计算机科学与技术学院 张岩

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—65
<p>2、串的链式存储</p> <p>例三：SUBSTR(S,m,n): 求串S从第m个字符开始到第n个字符为止的一个子串。(练习)</p> <p>3、串的成组链式存储</p> <ul style="list-style-type: none"> ▪ 删除操作 ▪ 插入操作 ▪ 插入和删除时，无用字符的处理 		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—66
<p>3.7 数组(ARRAY)</p> <p>3.7.1 抽象数据型数组</p> <ul style="list-style-type: none"> ● 数组是由下标(index)和值(value)组成的序对(index, value)的集合。 ● 也可以定义为是由相同类型的数据元素组成有限序列。 ● 数组在内存中是采用一组连续的地址空间存储的，正是由于此种原因，才可以实现下标运算。 ● 所有数组都是一个一维向量。 <p>数组1: (a₁, a₂, a₃, ..., a_i, ..., a_n);</p> <p>数组2: (a₁₁, a₁₂, a₂₁, ..., a_{2n}, ..., a_{ij}, ..., a_{m1}, ..., a_{mn}); 1≤i≤m, 1≤j≤n;</p> <p>数组3: (a₁₁, ..., a_{1n}, a₁₂₁, ..., a_{12n}, ..., a_{ijk}, ..., a_{mni}, ..., a_{mnp}); 1≤i≤m, 1≤j≤n, 1≤k≤p;</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—67
<p>3.7.1 抽象数据型数组</p> <p>n维数组中含有 $\prod_{i=1}^n b_i$ 个数据元素, 每个元素都受着 n 个关系的约束。在每个关系中, 元素 $a_{j_1 j_2 \dots j_n}$ ($0 \leq j_i \leq b_i - 2$) 都有一个直接后继元素。</p> <p>因此, 数组仍是一种特殊形式的线性表。对二维数组可以理解成一维数组, 其每个元素又是一个一维数组; 以此类推, 所谓 n 维数组同样如此。</p> <p>操作: CREATE (); 建立一个空数组 RETRIEVE (array, index); 返回第 index 个元素 STORE (array, index, value); 在数组array中, 为第index个元素赋值value 注: 由于高级语言中都提供了数组, 本课略去操作。</p>		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—68																								
<p>3.7.2 数组的实现</p> <p>1、数组的顺序存储</p> <p>数组的顺序表示, 指的是在计算机中, 用一组连续的存储单元来实现数组的存储。目前的高级程序设计语言都是这样实现的。</p> <p>两种存储方式:</p> <ul style="list-style-type: none"> 一是按行存储 (C语言、PASCAL等) 二是按列存储 (FORTRAN等) <p>elementtype A[2][3] :</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">第一行</td> <td style="border: none;"> <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>A[0][0]</td><td>A[0][1]</td><td>A[0][2]</td></tr> <tr><td>A[0][1]</td><td>A[1][0]</td><td>A[1][1]</td></tr> <tr><td>A[0][2]</td><td>A[1][1]</td><td>A[2][0]</td></tr> </table> </td> <td style="border: none;"></td> </tr> <tr> <td style="text-align: center;">第二行</td> <td style="border: none;"> <table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>A[0][1]</td><td>A[1][0]</td><td>A[2][1]</td></tr> <tr><td>A[1][1]</td><td>A[2][0]</td><td>A[3][0]</td></tr> <tr><td>A[2][2]</td><td>A[3][1]</td><td>A[4][0]</td></tr> </table> </td> <td style="border: none;"></td> </tr> </table> <p style="text-align: right; margin-top: -20px;">A = [A[0][0] A[0][1] A[0][2]] A[1][0] A[1][1] A[1][2]]</p> <p style="text-align: right; margin-top: -20px;">第一列 第二列 第三列</p>			第一行	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>A[0][0]</td><td>A[0][1]</td><td>A[0][2]</td></tr> <tr><td>A[0][1]</td><td>A[1][0]</td><td>A[1][1]</td></tr> <tr><td>A[0][2]</td><td>A[1][1]</td><td>A[2][0]</td></tr> </table>	A[0][0]	A[0][1]	A[0][2]	A[0][1]	A[1][0]	A[1][1]	A[0][2]	A[1][1]	A[2][0]		第二行	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>A[0][1]</td><td>A[1][0]</td><td>A[2][1]</td></tr> <tr><td>A[1][1]</td><td>A[2][0]</td><td>A[3][0]</td></tr> <tr><td>A[2][2]</td><td>A[3][1]</td><td>A[4][0]</td></tr> </table>	A[0][1]	A[1][0]	A[2][1]	A[1][1]	A[2][0]	A[3][0]	A[2][2]	A[3][1]	A[4][0]	
第一行	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>A[0][0]</td><td>A[0][1]</td><td>A[0][2]</td></tr> <tr><td>A[0][1]</td><td>A[1][0]</td><td>A[1][1]</td></tr> <tr><td>A[0][2]</td><td>A[1][1]</td><td>A[2][0]</td></tr> </table>	A[0][0]	A[0][1]	A[0][2]	A[0][1]	A[1][0]	A[1][1]	A[0][2]	A[1][1]	A[2][0]																
A[0][0]	A[0][1]	A[0][2]																								
A[0][1]	A[1][0]	A[1][1]																								
A[0][2]	A[1][1]	A[2][0]																								
第二行	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr><td>A[0][1]</td><td>A[1][0]</td><td>A[2][1]</td></tr> <tr><td>A[1][1]</td><td>A[2][0]</td><td>A[3][0]</td></tr> <tr><td>A[2][2]</td><td>A[3][1]</td><td>A[4][0]</td></tr> </table>	A[0][1]	A[1][0]	A[2][1]	A[1][1]	A[2][0]	A[3][0]	A[2][2]	A[3][1]	A[4][0]																
A[0][1]	A[1][0]	A[2][1]																								
A[1][1]	A[2][0]	A[3][0]																								
A[2][2]	A[3][1]	A[4][0]																								
哈尔滨工业大学计算机科学与技术学院 张岩																										

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—69
1、数组的顺序存储		
对二维数组有： $LOC(A[i][j]) = LOC(A[0][0]) + n \cdot i + j, 0 \leq i \leq m-1, 0 \leq j \leq n-1$		
对三维数组有： $LOC(A[i_1][i_2][i_3]) = LOC(A[0][0][0]) + d_2 \cdot d_3 \cdot i_1 + d_3 \cdot i_2 + i_3, 0 \leq i_1 \leq d_1-1, 0 \leq i_2 \leq d_2-1, 0 \leq i_3 \leq d_3-1$		
同理对n维数组有： $LOC(A[i_1][i_2] \cdots [i_n]) = LOC(A[0][0] \cdots [0]) + d_2 \cdot d_3 \cdots i_1 + d_3 \cdot d_4 \cdots d_n \cdot i_2 + d_n \cdot i_{n-1} + i_n$		
或： $LOC(A[i_1][i_2] \cdots [i_n]) = LOC(A[0][0] \cdots [0]) + \sum_{r=1}^n a_r \cdot i_r$		
而： $\begin{cases} \prod_{a=r+1}^n d_a & (1 \leq r \leq n-1) \\ a_n = 1 & 0 \leq i_1 \leq d_1-1, 0 \leq i_2 \leq d_2-1, \dots, 0 \leq i_n \leq d_n-1 \end{cases}$		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—70
2、数组的压缩存储		
(1) 特殊矩阵		
若n阶矩阵A中的元素满足下述性质 $a_{ij} = a_{ji}, 1 \leq i, j \leq n$ 则称n阶对称阵。		
对于对称矩阵，为实现节约存储空间，我们可以为每一对对称元素分配一个存储空间，这样，原来需要的 n^2 个元素压缩存储到 $n(n+1)/2$ 个元素空间。		
对称关系： 设 $sa[0, n(n+1)/2]$ 做为n阶对称阵A的存储结构，则 $sa[k]$ 和 a_{ij} 的一一对应关系为：		
$k = \begin{cases} i(i-1)/2 + j & \text{当 } i \geq j \\ j(j-1)/2 + i & \text{当 } i < j \end{cases}$		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—71
(1) 特殊矩阵		
映像关系： $A = [a_{11} a_{21} a_{22} a_{31} a_{32} a_{33} \dots a_{ii} \dots a_{n1} \dots a_{nn}]$		
$k = 1, 2, 3, 4, 5, 6, \dots, i(i-1)/2 + j, \dots, n(n-1)/2 + 1$ 对 $(n+1)/2$ 下三角矩阵可采用同样的方法。		
(2) 对角（带状）矩阵		
所有非零元素都集中在以主对角线为中心的带状区域内。 如图所示， $s=2$ ，称s为带宽，为实现压缩存储，我们只存储带区内的非零元素。不难总结出： $LOC[i, j] = LOC[1, 1] + [(2s+1)*(i-1)+(j-i)]*L$ $L = 1;$		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—72																																																						
(3) 稀疏矩阵																																																								
稀疏矩阵中，零元素的个数远远多于非零元素的个数。为实现压缩存储，我们仍考虑只存储非零元素。																																																								
$M = \begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{pmatrix} T = M' = \begin{pmatrix} 0 & 0 & -3 & 0 & 0 & 15 \\ 12 & 0 & 0 & 0 & 18 & 0 \\ 9 & 0 & 0 & 24 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$																																																								
<table border="1"> <thead> <tr> <th>i</th> <th>j</th> <th>v</th> <th>i</th> <th>j</th> <th>v</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>2</td> <td>12</td> <td>1</td> <td>3</td> <td>-3</td> </tr> <tr> <td>1</td> <td>3</td> <td>9</td> <td>1</td> <td>6</td> <td>15</td> </tr> <tr> <td>3</td> <td>1</td> <td>-3</td> <td>2</td> <td>1</td> <td>12</td> </tr> <tr> <td>4</td> <td>6</td> <td>14</td> <td>2</td> <td>5</td> <td>18</td> </tr> <tr> <td>4</td> <td>3</td> <td>24</td> <td>3</td> <td>1</td> <td>9</td> </tr> <tr> <td>5</td> <td>2</td> <td>18</td> <td>3</td> <td>4</td> <td>24</td> </tr> <tr> <td>6</td> <td>1</td> <td>15</td> <td>4</td> <td>6</td> <td>-7</td> </tr> <tr> <td>6</td> <td>4</td> <td>-7</td> <td>6</td> <td>3</td> <td>14</td> </tr> </tbody> </table> <pre>#define MAXSIZE 12500 typedef struct { int i, j; elementtype e; } Triple; typedef struct { Triple data[MAXSIZE+1]; int mu, nu, tu; } TSMatrix;</pre>			i	j	v	i	j	v	1	2	12	1	3	-3	1	3	9	1	6	15	3	1	-3	2	1	12	4	6	14	2	5	18	4	3	24	3	1	9	5	2	18	3	4	24	6	1	15	4	6	-7	6	4	-7	6	3	14
i	j	v	i	j	v																																																			
1	2	12	1	3	-3																																																			
1	3	9	1	6	15																																																			
3	1	-3	2	1	12																																																			
4	6	14	2	5	18																																																			
4	3	24	3	1	9																																																			
5	2	18	3	4	24																																																			
6	1	15	4	6	-7																																																			
6	4	-7	6	3	14																																																			
哈尔滨工业大学计算机科学与技术学院 张岩																																																								

哈工大计算机考研全套视频和资料，真题、考点、典型题、命题规律独家视频讲解！
详见：网学天地（www.e-studysky.com）；咨询QQ：2696670126

数据结构与算法	第三章 线性表	3—73
(3) 稀疏矩阵		
由稀疏矩阵的三元组表示求转置矩阵TransposMatrix； Void TransposMatrix(TSMATRIX M, TSMATRIX &T) { T.mu = M.nu; T.nu = M.mu; T.tu = M.tu; if (T.nu) { q = 1; for (col = 1; col <= M.nu; ++col) for (p = 1; p <= M.tu; ++p) { if (M.data[p].j == col) { T.data[q].i = M.data[p].j; T.data[q].j = M.data[p].i; T.data[q].e = M.data[p].e; ++q; } } } 分析: $T(n) = O(nu \cdot tu)$		

数据结构与算法	第三章 线性表	3—75
(3) 稀疏矩阵		
void FastTransposMatrix(TSMATRIX M, TSMATRIX &T) { T.mu = M.nu; T.nu = M.mu; T.tu = M.tu; if (T.nu) { for (col = 1; col <= M.nu; ++col) num[col] = 0; for (t = 1; t <= M.tu; ++t) ++num[M.data[t].j]; cpot[1] = 1; for (col = 2; col < M.nu; col++) cpot[col] = cpot[col - 1] + num[col - 1]; for (p = 1; p <= M.tu; ++p) { col = M.data[p].j; q = cpot[col]; T.data[q].i = M.data[p].j; T.data[q].j = M.data[p].i; T.data[q].e = M.data[p].e; ++cpot[col]; } } } 转置的改进算法: $T(n) = O(nu + tu)$		

数据结构与算法	第三章 线性表	3—74																								
(3) 稀疏矩阵																										
算法改进： 设向量 $num[col]$ 表示矩阵M中第 col 列中非零元素的个数, $cpot[col]$ 指示M中第 col 列的第一个非零元素在 b 中的恰当位置。 $\{ cpot[1] = 1;$ $\{ cpot[col] = cpot[col - 1] + num[col - 1] \quad 2 \leq col \leq a.nu$																										
矩阵M的向量 $cpot$ 的值 <table border="1"><tr><th>col</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr><tr><th>num[col]</th><td>2</td><td>2</td><td>2</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><th>cpot[col]</th><td>1</td><td>3</td><td>5</td><td>7</td><td>8</td><td>8</td><td>9</td></tr></table>			col	1	2	3	4	5	6	7	num[col]	2	2	2	1	0	1	0	cpot[col]	1	3	5	7	8	8	9
col	1	2	3	4	5	6	7																			
num[col]	2	2	2	1	0	1	0																			
cpot[col]	1	3	5	7	8	8	9																			

数据结构与算法	第三章 线性表	3—76
3、数组的链接式存储		
多维数组,特别是稀疏矩阵可以采用链接式存储。 结点形式: 结点类型: <pre>struct node { node *LEFT, *UP; int ROW, COL; valuetype VAL; };</pre>		

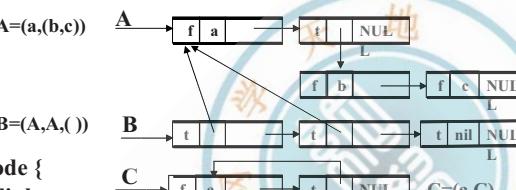
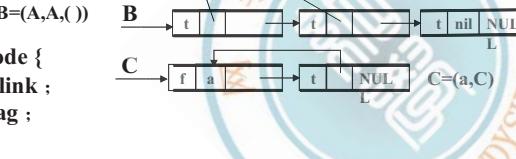
二维数组(矩阵)的链接表示,见p90。

数据结构与算法	第三章 线性表	3—77
3.8 广义表 (General Lists)		
广义表示线性表的一种推广结构，线性表要求是相同的元素类型，而广义表中的元素可以取不同类型，可以是最基本的不可再分割的原子，也可以是广义表本身。广义表是由零个原子，或若干个原子或若干个广义表组成的有穷序列。		
通常将广义表表示为： $A = (a_1, a_2, \dots, a_n)$ 其中，A是名称，元素个数n是表的长度；若 a_i 不是原子，则称其为A的子表。 注意广义表的递归性。		

例如：
 $A = (a, (b, a, b), (), c, ((2)))$;
 $B = ()$;
 $C = (e)$;
 $D = (A, B, C)$;
 $E = (a, E)$;

哈尔滨工业大学计算机科学与技术学院 张岩

数据结构与算法	第三章 线性表	3—78
3.8 广义表 (Lists)		
结论： ① 广义表的元素可以是子表，子表的元素还可以是子表，……，广义表是一个多层次的结构； ② 一个广义表可以被其他广义表所共享。 ③ 广义表是一个递归的表，即广义表可以是其本身的子表。		
操作： ① CAR (L) ；返回广义表 L 的第一个元素 ② CDR (L) ；返回广义表 L 除第一个元素以外的所有元素 ③ APPEND (L, M) ；返回广义表 L + M ④ EQUAL (L, M) ；判广义表 L 和 M 是否相等 ⑤ LENGTH (L) ；求广义表 L 的长度		
哈尔滨工业大学计算机科学与技术学院 张岩		

数据结构与算法	第三章 线性表	3—79
广义表的存储结构		
<p>$A=(a,(b,c))$ A → [f a] → [t] → NUL </p> <p>$B=(A,A,())$ B → [t] → [t] → [t nil] → NUL </p> <pre> struct listnode { listnode *link; boolean tag; union { char data; listnode *dlink; }; }; typedef listnode *listpointer; </pre>		

数据结构与算法	第三章 线性表	3—80
广义表的操作		
<pre> boolean EQUAL(listpointer S, T) { boolean x, y; y = FALSE; if ((S == NULL) && (T == NULL)) y = TRUE; else if ((S != NULL) && (T != NULL)) if (S->tag == T->tag) { if (S->tag == FALSE) { if (S->element.data == T->element.data) x = TRUE; else x = FALSE; } else x = EQUAL(S->element.data, T->element.data); if (x == TRUE) y = EQUAL(S->link, T->link); } return y; } </pre>		