



“十二五”普通高等教育本科国家级规划教材

高等学校计算机专业核心课

名师精品·系列教材

数据结构

(C语言版 第3版)

双色版 | 附微课视频

李冬梅 严蔚敏 吴伟民 编著

DATA STRUCTURES IN C
(3RD EDITION)

符合最新统考考研大纲要求，紧跟最新技术发展
配套立体化教学资源，支持线上线下混合式教学
案例驱动，算法清晰，例题、习题类型丰富



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS



“十二五”普通高等教育本科国家级规划教材

高等学校计算机专业核心课

名师精品·系列教材

数据结构

(C语言版 第3版)

双色版 | 附微课视频

李冬梅 严蔚敏 吴伟民 编著

DATA STRUCTURES IN C
(3RD EDITION)

人民邮电出版社
北京

图书在版编目 (CIP) 数据

数据结构 : C 语言版 : 双色版 / 李冬梅, 严蔚敏, 吴伟民编著. — 3 版. — 北京 : 人民邮电出版社, 2024. — (高等学校计算机专业核心课名师精品系列教材). — ISBN 978-7-115-65125-9

I. TP311.12; TP312.8

中国国家版本馆 CIP 数据核字第 2024AN0707 号

内 容 提 要

本书在选材与编排上, 贴近当前普通高等院校“数据结构”课程的现状和发展趋势, 符合最新的全国硕士研究生统一入学考试计算机科学与技术学科数据结构考试大纲, 内容难度适中, 突出实用性和应用性。全书共 8 章, 内容包括绪论, 线性表, 栈和队列, 串、数组和广义表, 树和二叉树, 图, 查找和排序。全书采用类 C 语言作为数据结构和算法的描述语言。

本书可作为普通高等院校计算机和信息技术相关专业“数据结构”课程的教材, 也可供从事计算机工程与应用工作的科技工作者参考。

-
- ◆ 编 著 李冬梅 严蔚敏 吴伟民
责任编辑 刘 博
责任印制 陈 桦
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <https://www.ptpress.com.cn>
大厂回族自治县聚鑫印刷有限责任公司印刷
 - ◆ 开本: 787×1092 1/16
印张: 21.25 2024 年 8 月第 3 版
字数: 560 千字 2026 年 8 月河北第 9 次印刷
-

定价: 59.80 元

读者服务热线: (010)81055256 印装质量热线: (010)81055316
反盗版热线: (010)81055315



第1章 绪论 1

1.1 数据结构的研究内容	1
1.2 数据结构的基本概念和术语	3
1.2.1 数据、数据元素、数据项和数据对象	3
1.2.2 数据结构	4
1.2.3 数据类型和抽象数据类型	6
1.3 抽象数据类型的表示与实现	7
1.4 算法和算法分析	10
1.4.1 算法的定义及特性	11
1.4.2 评价算法优劣的基本标准	11
1.4.3 算法的时间复杂度	11
1.4.4 算法的空间复杂度	15
1.5 小结	16
习题	17

第2章 线性表 19

2.1 线性表的定义和特点	19
2.2 案例引入	20
2.3 线性表的类型定义	22
2.4 线性表的顺序表示和实现	23
2.4.1 线性表的顺序表示	23
2.4.2 顺序表中基本操作的实现	25
2.5 线性表的链式表示和实现	29
2.5.1 单链表的定义和表示	29
2.5.2 单链表基本操作的实现	32
2.5.3 循环链表	38

2.5.4 双向链表	38
2.6 顺序表和链表的比较	40
2.6.1 空间性能的比较	40
2.6.2 时间性能的比较	40
2.7 线性表的应用	41
2.7.1 线性表的合并	41
2.7.2 有序表的合并	42
2.8 案例分析与实现	44
2.9 LeetCode 算法练习题	49
2.10 小结	58
习题	59

第3章 栈和队列 62

3.1 栈和队列的定义和特点	62
3.1.1 栈的定义和特点	62
3.1.2 队列的定义和特点	63
3.2 案例引入	63
3.3 栈的表示和实现	65
3.3.1 栈的类型定义	65
3.3.2 顺序栈的表示和实现	66
3.3.3 链栈的表示和实现	68
3.4 栈与递归	70
3.4.1 采用递归算法解决的问题	70
3.4.2 递归过程与递归工作栈	74
3.4.3 递归算法的效率分析	76
3.4.4 利用栈将递归转换为非递归的方法	76



3.5 队列的表示和实现	77
3.5.1 队列的类型定义	77
3.5.2 顺序队的表示和实现	78
3.5.3 链队的表示和实现	81
3.6 案例分析与实现	83
3.7 LeetCode 算法练习题	90
3.8 小结	101
习题	102

第4章 串、数组和广义表 105

4.1 串的定义	105
4.2 案例引入	106
4.3 串的类型定义、存储结构及其运算	107
4.3.1 串的抽象类型定义	107
4.3.2 串的存储结构	108
4.3.3 串的模式匹配算法	109
4.4 数组	118
4.4.1 数组的类型定义	118
4.4.2 数组的顺序存储	119
4.4.3 特殊矩阵的压缩存储	120
4.5 广义表	122
4.5.1 广义表的定义	122
4.5.2 广义表的存储结构	123
4.6 案例分析与实现	125
4.7 LeetCode 算法练习题	126
4.8 小结	134
习题	134

第5章 树和二叉树 137

5.1 树和二叉树的定义	137
5.1.1 树的定义	137
5.1.2 树的基本术语	138
5.1.3 二叉树的定义	139
5.2 案例引入	140
5.3 树和二叉树的抽象数据类型定义	141
5.4 二叉树的性质和存储结构	144
5.4.1 二叉树的性质	144
5.4.2 二叉树的存储结构	146
5.5 遍历二叉树和线索二叉树	147
5.5.1 遍历二叉树	147
5.5.2 线索二叉树	154
5.6 树和森林	158
5.6.1 树的存储结构	158
5.6.2 森林与二叉树的转换	160
5.6.3 树和森林的遍历	161
5.7 哈夫曼树及其应用	162
5.7.1 哈夫曼树的基本概念	162
5.7.2 哈夫曼树的构造算法	163
5.7.3 哈夫曼编码	165
5.8 并查集及其应用	168
5.8.1 并查集的基本概念	168
5.8.2 并查集的实现及优化	169
5.9 案例分析与实现	173
5.10 LeetCode 算法练习题	176
5.11 小结	180
习题	181

**第6章 图 184**

6.1 图的定义和基本术语	184
6.1.1 图的定义	184
6.1.2 图的基本术语	185
6.2 案例引入	186
6.3 图的类型定义	187
6.4 图的存储结构	188
6.4.1 邻接矩阵	189
6.4.2 邻接表	191
6.4.3 十字链表	193
6.4.4 邻接多重表	194
6.5 图的遍历	196
6.5.1 深度优先搜索	196
6.5.2 广度优先搜索	198
6.6 图的应用	200
6.6.1 最小生成树	200
6.6.2 最短路径	205
6.6.3 拓扑排序	211
6.6.4 关键路径	214
6.7 案例分析与实现	219
6.8 LeetCode 算法练习题	221
6.9 小结	225
习题	227

第7章 查找 230

7.1 查找的基本概念	230
7.2 线性表的查找	231
7.2.1 顺序查找	231

7.2.2 折半查找	232
7.2.3 分块查找	235
7.3 树表的查找	237
7.3.1 二叉排序树	237
7.3.2 平衡二叉树	244
7.3.3 红黑树	249
7.3.4 B- 树	259
7.3.5 B+ 树	267
7.4 散列表的查找	268
7.4.1 散列表的基本概念	268
7.4.2 散列函数的构造方法	270
7.4.3 处理冲突的方法	272
7.4.4 散列表的查找	274
7.5 LeetCode 算法题	277
7.6 小结	284
习题	286

第8章 排序 289

8.1 基本概念和排序方法概述	289
8.1.1 排序的基本概念	289
8.1.2 内部排序方法的分类	290
8.1.3 待排序记录的存储方式	291
8.1.4 排序算法效率的评价指标	291
8.2 插入排序	291
8.2.1 直接插入排序	292
8.2.2 折半插入排序	293
8.2.3 希尔排序	294
8.3 交换排序	296



8.3.1 冒泡排序	296	8.7 外部排序	314
8.3.2 快速排序	298	8.7.1 外部排序的基本方法	315
8.4 选择排序	301	8.7.2 多路平衡归并的实现	316
8.4.1 简单选择排序	301	8.7.3 置换 - 选择排序	317
8.4.2 树形选择排序	303	8.7.4 最佳归并树	321
8.4.3 堆排序	304	8.8 LeetCode 算法题	322
8.5 归并排序	309	8.9 小结	326
8.6 基数排序	311	习题	328
8.6.1 多关键字的排序	311		
8.6.2 链式基数排序	311		
		参考文献	330

度 n 为 0 的空表时, L 指针为空 (判定空表的条件可记为: $L == \text{NULL}$)。

增加头结点后, 无论链表是否为空, 头指针都是指向头结点的非空指针。如图 2.10 (a) 所示的非空单链表, 头指针指向头结点。若为空表, 则头结点的指针域为空 (判定空表的条件记为: $L \rightarrow \text{next} == \text{NULL}$), 如图 2.10 (b) 所示。

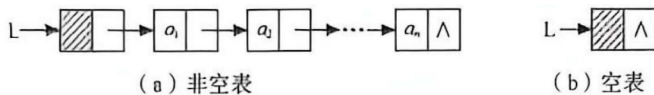


图 2.10 带头结点的单链表

在顺序表中, 由于逻辑上相邻的两个元素在物理位置上紧邻, 则每个元素的存储位置都从线性表的起始位置计算得到。而在单链表中, 各个元素的存储位置都是随意的。然而, 每元素的存储位置都包含在其直接前驱结点的信息之中。假设 p 是指向单链表中第 i 个数据元素 (结点 a_i , 即数据域为 a_i 的结点) 的指针, 则 $p \rightarrow \text{next}$ 是指向第 $i+1$ 个数据元素 (结点 a_{i+1}) 的指针。换句话说, 若 $p \rightarrow \text{data} = a_i$, 则 $p \rightarrow \text{next} \rightarrow \text{data} = a_{i+1}$ 。由此, 单链表是非随机存储的存储结构, 要取得第 i 个数据元素必须从头指针出发顺链进行寻找, 也称为顺序存储的存储结构。因此, 其基本操作的实现不同于顺序表。

2.5.2 单链表基本操作的实现

1. 初始化

单链表的初始化操作就是构造一个如图 2.10 (b) 所示的空表。

算法 2.6 单链表的初始化

【算法步骤】

- ① 生成新结点作为头结点, 用头指针 L 指向头结点。
- ② 头结点的指针域置空。

【算法描述】

```
Status InitList(LinkList &L)
{ // 构造一个空的单链表 L
    L = new LNode;           // 生成新结点作为头结点, 用头指针 L 指向头结点
    L->next = NULL;          // 头结点的指针域置空
    return OK;
}
```



2. 取值

和顺序表不同, 链表中逻辑相邻的结点并没有存储在物理相邻的单元中, 这样, 根据给定的结点位置序号 i , 在链表中获取该结点的值不能像顺序表那样随机访问, 而只能从链表的首元结点出发, 顺着链域 next 逐个结点向下访问。

算法 2.7 单链表的取值

【算法步骤】

- ① 用指针 p 指向首元结点, 用 j 做计数器初值赋为 1。
- ② 从首元结点开始依次顺着链域 next 向下访问, 只要指向当前结点的指针 p 不为空 (NULL), 并且没有到达序号为 i 的结点, 则循环执行以下操作:
 - p 指向下一个结点;
 - 计数器 j 相应加 1。



③ 退出循环时, 如果指针 p 为空, 或者计数器 j 大于 i , 说明指定的序号 i 值不合法 (i 大于表长 n 或 i 小于等于 0), 取值失败返回 ERROR; 否则取值成功, 此时 $j = i$ 时, p 所指的结点就是要找的第 i 个结点, 用参数 e 保存当前结点的数据域, 返回 OK。

【算法描述】

```
Status GetElem(LinkList L, int i, ElemType &e)
{
    // 在带头结点的单链表 L 中根据序号 i 获取元素的值, 用 e 返回 L 中第 i 个数据元素的值
    p=L->next; j=1; // 初始化, p 指向首元结点, 计数器 j 初值赋为 1
    while (p && j<i) // 顺链域向后查找, 直到 p 为空或 p 指向第 i 个元素
    {
        p=p->next; // p 指向下一个结点
        ++j; // 计数器 j 相应加 1
    }
    if (!p || j>i) return ERROR; // i 值不合法 i>n 或 i<=0
    e=p->data; // 取第 i 个结点的数据域
    return OK;
}
```

【算法分析】

该算法的基本操作是比较 j 和 i 并后移指针 p , while 循环体中的语句频度与位置 i 有关。若 $1 \leq i \leq n$, 则频度为 $i-1$, 一定能取值成功; 若 $i > n$, 则频度为 n , 取值失败。因此算法 2.7 的最坏时间复杂度为 $O(n)$ 。

假设每个位置上元素的取值概率相等, 即:

$$p_i = 1/n$$

则:

$$ASL = \frac{1}{n} \sum_{i=1}^n (i-1) = \frac{n-1}{2} \quad (2-9)$$

由此可见, 单链表取值算法的平均时间复杂度为 $O(n)$ 。

3. 查找

链表中按值查找的过程和顺序表类似, 从链表的首元结点出发, 依次将结点值和给定值 e 进行比较, 返回查找结果。

算法 2.8 单链表的查找

【算法步骤】

- ① 用指针 p 指向首元结点。
- ② 从首元结点开始依次顺着链域 $next$ 向下查找, 只要指向当前结点的指针 p 不为空, 并且 p 所指结点的数据域不等于给定值 e , 则循环执行以下操作: p 指向下一个结点。

- ③ 返回 p 。若查找成功, p 此时指向结点的地址值, 若查找失败, 则 p 的值为 NULL。

【算法描述】

```
LNode *LocateElem(LinkList L, ElemType e)
{
    // 在带头结点的单链表 L 中查找值为 e 的元素
    p=L->next; // 初始化, p 指向首元结点
    while (p && p->data!=e) // 顺链域向后查找, 直到 p 为空或 p 所指结点的数据域等于 e
    {
        p=p->next; // p 指向下一个结点
    }
    return p; // 查找成功返回值为 e 的结点地址 p, 查找失败 p 为 NULL
}
```



单链表的查找

【算法分析】

该算法的执行时间与待查找的值 c 相关, 其平均时间复杂度分析类似于算法 2.7, 也为 $O(n)$ 。

4. 插入

假设要在单链表的两个数据元素 a 和 b 之间插入一个数据元素 x , 已知 p 为其单链表存储结构中指向结点 a 的指针, 如图 2.11 (a) 所示。

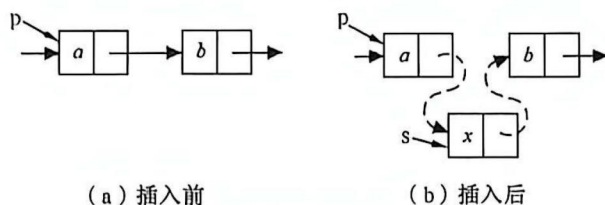


图 2.11 在单链表中插入结点时指针变化状况

为插入数据元素 x , 首先要生成一个数据域为 x 的结点, 然后将之插入单链表中。根据插入的逻辑定义, 还需要修改结点 a 中的指针域, 令其指向结点 x , 而结点 x 中的指针域应指向结点 b , 从而实现 3 个元素 a 、 b 和 x 之间逻辑关系的变化。插入后的单链表如图 2.11 (b) 所示。假设 s 为指向结点 x 的指针, 则上述指针修改用语句描述即为:

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

算法 2.9 单链表的插入

【算法步骤】

将值为 e 的新结点插入表的第 i 个结点的位置, 即插入结点 a_{i-1} 与 a_i 之间, 具体插入过程如图 2.12 所示, 图中对应的 5 个步骤说明如下。

- ① 查找结点 a_{i-1} 并由指针 p 指向该结点。
- ② 生成一个新结点 $*s$ 。
- ③ 将新结点 $*s$ 的数据域置为 e 。
- ④ 将新结点 $*s$ 的指针域指向结点 a_i 。
- ⑤ 将结点 $*p$ 的指针域指向新结点 $*s$ 。

【算法描述】

```
Status ListInsert(LinkList &L, int i, ElemType e)
{ // 在带头结点的单链表 L 中第 i 个位置插入值为 e 的新结点
    p=L; j=0;
    while(p && (j<i-1))
        (p=p->next; ++j); // 查找第 i-1 个结点, p 指向该结点
    if(!p || j>i-1) return ERROR; // i > n+1 或者 i < 1
    s=new LNode; // 生成新结点 *s
    s->data=e; // 将结点 *s 的数据域置为 e
    s->next=p->next; // 将结点 *s 的指针域指向结点 ai
    p->next=s; // 将结点 *p 的指针域指向结点 *s
    return OK;
}
```



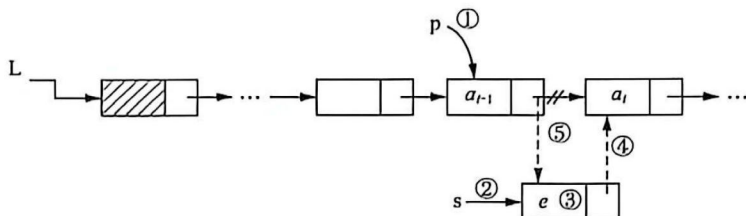
单链表的插入

说明

和顺序表一样, 如果表中有 n 个结点, 则插入操作中合法的插入位置有 $n+1$ 个, 即 $1 \leq i \leq n+1$ 。当 $i = n+1$ 时, 新结点则插在链表尾部。

【算法分析】

单链表的插入操作虽然不需要像顺序表的插入操作那样移动元素，但平均时间复杂度仍为 $O(n)$ 。这是因为，为了在第 i 个结点之前插入一个新结点，必须首先找到第 $i-1$ 个结点，其时间复杂度与算法 2.7 相同，为 $O(n)$ 。

图 2.12 在单链表第 i 个位置上插入新结点的过程

5. 删除

要删除单链表中指定位置的元素，同插入元素一样，首先应该找到该位置的前驱结点。如图 2.13 所示，在单链表中删除元素 b 时，应该首先找到其前驱结点 a 。为了在单链表中实现元素 a 、 b 和 c 之间逻辑关系的变化，仅需修改结点 a 中的指针域即可。假设 p 为指向结点 a 的指针，则修改指针的语句为：

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

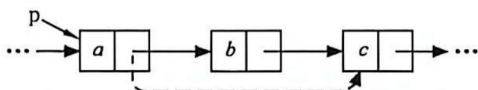


图 2.13 在单链表中删除结点时指针的变化

但在删除结点 b 时，除了修改结点 a 的指针域外，还要释放结点 b 所占的空间，所以在修改指针前，应该引入另一指针 q ，临时保存结点 b 的地址以备释放。

算法 2.10 单链表的删除

【算法步骤】

删除单链表的第 i 个结点 a_i 的具体过程如图 2.14 所示，图中对应的 4 个步骤说明如下。

- ① 查找结点 a_{i-1} 并由指针 p 指向该结点。
- ② 临时保存待删除结点 a_i 的地址在 q 中，以备释放。
- ③ 将结点 $*p$ 的指针域指向 a_i 的直接后继结点。
- ④ 释放结点 a_i 的空间。

【算法描述】

```

Status ListDelete(LinkList &L, int i)
{
    // 在带头结点的单链表 L 中，删除第 i 个元素
    p=L; j=0;
    while((p->next) && (j<i-1))          // 查找第 i-1 个结点，p 指向该结点
        (p=p->next; ++j;);
    if(! (p->next) || (j>i-1)) return ERROR; // 当 i>n 或 i<1 时，删除位置不合理
    q=p->next;                             // 临时保存被删结点的地址以备释放
    p->next=q->next;                         // 改变删除结点前驱结点的指针域
    delete q;                              // 释放删除结点的空间
    return OK;
}

```



单链表的删除

说明

删除算法中的循环条件 ($p \rightarrow \text{next} \& \& j < i - 1$) 和插入算法中的循环条件 ($p \& \& (j < i - 1)$) 是有所区别的。因为插入操作中合法的插入位置有 $n + 1$ 个, 而删除操作中合法的删除位置只有 n 个, 如果使用与插入操作相同的循环条件, 则会出现引用空指针的情况, 使删除操作失败。

【算法分析】

类似于插入算法, 删除算法的时间复杂度亦为 $O(n)$ 。

6. 创建单链表

算法 2.6 的初始化操作是创建一个只有一个头结点的空链表, 而

上面链表的其他算法都是假定链表已经存在多个结点。那么, 如何建立一个包括若干个结点的链表呢? 链表和顺序表不同, 它是一种动态结构。整个可用存储空间可为多个链表共同享用, 每个链表占用的空间不需预先分配划定, 而是由系统按需即时生成。因此, 建立线性表的存储结构的过程就是一个动态生成链表的过程。即从空表的初始状态起, 依次建立各元素结点并逐个插入链表。

根据结点插入位置的不同, 链表的创建方法可分为前插法和后插法。

(1) 前插法

前插法是通过将新结点逐个插入链表的头部 (头结点之后) 来创建链表, 每次申请一个结点, 读入相应的数据元素值, 然后将新结点插入到头结点之后。

算法 2.11 前插法创建单链表

【算法步骤】

- ① 创建一个只有头结点的空链表。
- ② 根据待创建链表包括的元素个数 n , 循环 n 次执行以下操作:
 - 生成一个新结点 $*p$;
 - 输入元素值赋给新结点 $*p$ 的数据域;
 - 将新结点 $*p$ 插入到头结点之后。

图 2.15 所示为线性表 (a, b, c, d, e) 前插法的创建过程, 因为每次插入在链表的头部, 应该逆位序输入数据, 依次输入 e, d, c, b, a , 输入顺序和线性表中的逻辑顺序是相反的。

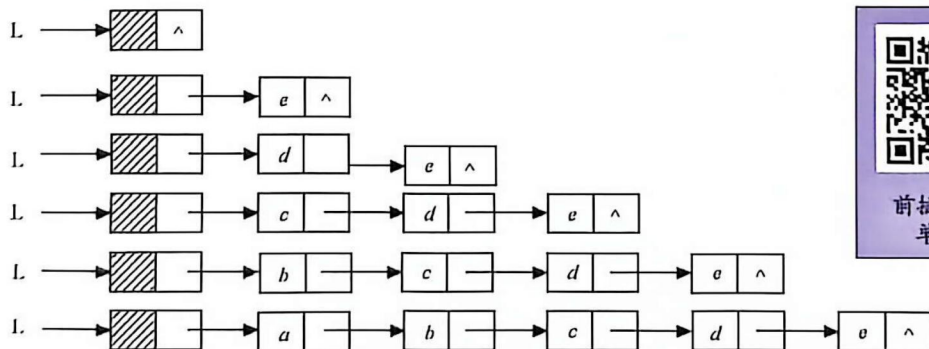


图 2.15 前插法创建单链表



前插法创建单链表

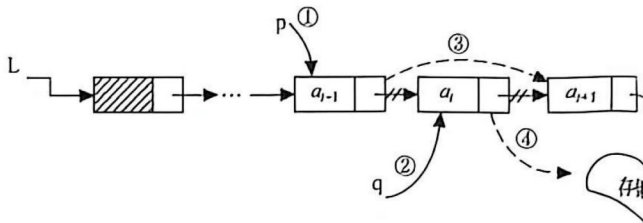


图 2.14 删除单链表第 i 个结点的过程

【算法描述】

```

void CreateList_H(LinkList &L, int n)
{ // 逆位序输入n个元素的值, 建立带头结点的单链表 L
  L=new LNode;
  L->next=NULL; // 先建立一个带头结点的空链表
  for (i=0; i<n; ++i)
  {
    p=new LNode; // 生成新结点 *p
    cin>>p->data; // 输入元素值赋给新结点 *p 的数据域
    p->next=L->next; L->next=p; // 将新结点 *p 插入到头结点之后
  }
}

```

显然, 算法 2.11 的时间复杂度为 $O(n)$ 。

(2) 后插法

后插法是通过将新结点逐个插入链表的尾部来创建链表。同前插法一样, 每次申请一个新结点, 读入相应的数据元素值。不同的是, 为了使新结点能够插入表尾, 需要增加一个尾指针 r 指向链表的尾结点。

算法 2.12 后插法创建单链表

【算法步骤】

- ① 创建一个只有头结点的空链表。
- ② 尾指针 r 初始化, 指向头结点。
- ③ 根据创建链表包括的元素个数 n , 循环 n 次执行以下操作:
 - 生成一个新结点 $*p$;
 - 输入元素值赋给新结点 $*p$ 的数据域;
 - 将新结点 $*p$ 插入尾结点 $*r$ 之后;
 - 尾指针 r 指向新的尾结点 $*p$ 。

图 2.16 所示为线性表 (a, b, c, d, e) 后插法的创建过程, 读入数据的顺序和线性表中的逻辑顺序是相同的。

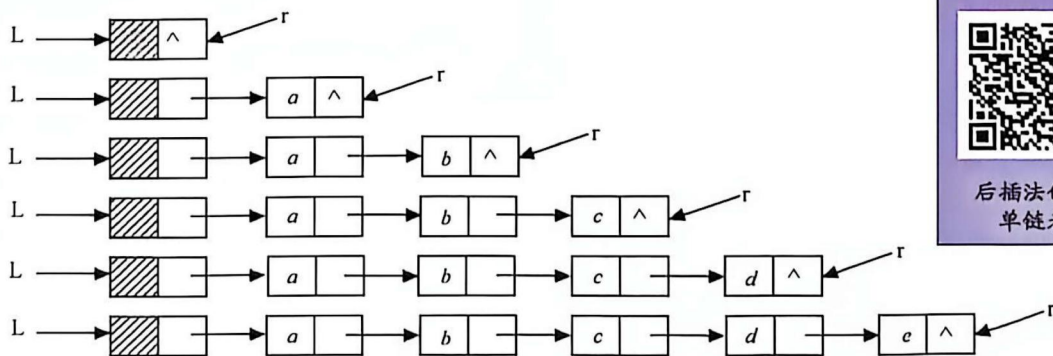


图 2.16 后插法创建单链表



后插法创建
单链表

【算法描述】

```

void CreateList_R(LinkList &L, int n)
{ // 正位序输入n个元素的值, 建立带头结点的单链表 L
  L=new LNode;
  L->next=NULL; // 先建立一个带头结点的空链表
  r=L; // 尾指针 r 指向头结点
  for (i=0; i<n; ++i)
  {
    p=new LNode;
    cin>>p->data;
    p->next=r->next;
    r->next=p;
  }
}

```

```

{
    p=new LNode;           // 生成新结点
    cin>>p->data;           // 输入元素值赋给新结点 *p 的数据域
    p->next=NULL; r->next=p; // 将新结点 *p 插入尾结点 *r 之后
    r=p;                   // r 指向新的尾结点 *p
}

```

算法 2.12 的时间复杂度亦为 $O(n)$ 。

2.5.3 循环链表

循环链表 (Circular Linked List) 是另一种形式的链式存储结构。其特点是表中最后结点的指针域指向头结点, 整个链表形成一个环。由此, 从表中任一结点出发均可找到表头结点, 图 2.17 所示为单链的循环链表。类似地, 还可以有多重链的循环链表。

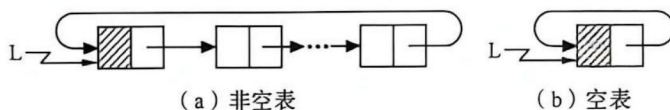


图 2.17 单循环链表

循环单链表的操作和单链表基本一致, 差别仅在于: 当链表遍历时, 判别当前指针指向表尾结点的终止条件不同。在单链表中, 判别条件为 $p \neq \text{NULL}$ 或 $p \rightarrow \text{next} \neq \text{NULL}$, 单链表的判别条件为 $p \neq L$ 或 $p \rightarrow \text{next} \neq L$ 。

在某些情况下, 若在循环链表中设立尾结点的尾指针而不设头指针 [见图 2.18 (a)] 使一些操作简化。例如, 将两个线性表合并成一个表时, 仅需将第一个表的尾结点的 next 指向第二个表的第一个结点, 第二个表的尾结点的指针域指向第一个表的头结点, 然后移两个表的头结点。当线性表以图 2.18 (a) 的循环链表作存储结构时, 这个操作仅需改变两值即可, 主要语句段如下:

```

p = B->next->next;
B->next = A->next;
A->next = p;

```

上述操作的时间复杂度为 $O(1)$, 合并后的表如图 2.18 (b) 所示。

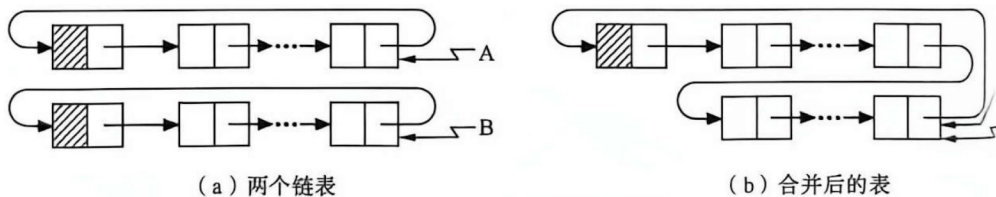


图 2.18 仅设尾指针的循环链表

2.5.4 双向链表

以上讨论的链式存储结构的结点中只有一个指示直接后继的指针域, 由此, 从某个结点只能顺指针向后寻查其他结点。若要寻查结点的直接前驱, 则必须从表头指针出发。说, 在单链表中, 查找直接后继的执行时间为 $O(1)$, 而查找直接前驱的执行时间为 $O(n)$ 。为克服单链表这种单向性的缺点, 可利用**双向链表** (Double Linked List)。

顾名思义, 在双向链表的结点中有两个指针域, 一个指向直接后继, 另一个指向直接前驱。

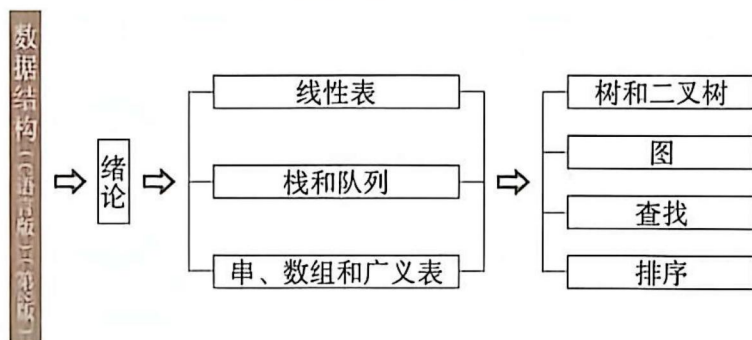
本 / 书 / 特 / 色

本书在选材与编排上, 贴近当前普通高等院校“数据结构”课程的现状和发展趋势, 符合最新的全国硕士研究生统一入学考试计算机科学与技术学科数据结构考试大纲, 内容难度适中, 突出实用性和应用性。全书共 8 章, 内容包括绪论, 线性表, 栈和队列, 串、数组和广义表, 树和二叉树, 图, 查找和排序。全书采用类 C 语言作为数据结构和算法的描述语言。

本书可作为普通高等院校计算机和信息技术相关专业“数据结构”课程的教材, 也可供从事计算机工程与应用工作的科技工作者参考。



本教材的结构框图



用书教师扫码下载
本书配套资源



教材服务热线: 010-81055256
反馈 / 投稿 / 推荐信箱: 315@ptpress.com.cn
人邮教育服务与资源下载社区: www.rjlaoyu.com



ISBN 978-7-115-65125-9



定价: 59.80元

封面设计: 董志桢