普通高等教育“十二五”创新型规划教材

数据结构综合设计实验教程

SHUJU JIEGOU ZONGHE SHEJI SHIYAN JIAOCHENG

主　编　刘城霞
副主编　蔡　英　吴　燕　田英爱

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

普通高等教育“十二五”创新型规划教材

数据结构综合设计实验教程

SHUJU JIEGOU ZONGHE SHEJI SHIYAN JIAOCHENG

主　编　刘城霞
副主编　蔡　英　吴　燕　田英爱
主　审　王铁峰

北京理工大学出版社
BEIJING INSTITUTE OF TECHNOLOGY PRESS

内 容 简 介

本书内容共分 8 章，前 4 章为基本数据结构的应用，第 5 章为多种数据结构的综合应用，第 6～8 章为数据结构知识扩展（部分数据结构书中含有这部分内容，但由于课时等原因讲解较少或未讲，因此可以放到综合设计中进行深入讲解；还有部分数据结构书中没有这部分内容，可以作为补充材料进行讲解）。本书在最后列出了一些实用的数据结构实践题目，可以方便老师在实践教学中选用或者学生自己选做。本书附录部分给出了实验报告的基本格式和实验报告范例，学生可以参照其要求进行实验。

本书可作为高等院校计算机专业及相关专业的教材或参考书，也可供从事软件开发工作的人员和计算机编程爱好者参考。

图书在版编目(CIP)数据

数据结构综合设计实验教程/刘城霞主编. —北京：北京理工大学出版社，2012. 9
ISBN 978－7－5640－6773－1

Ⅰ. ①数…　Ⅱ. ①刘…　Ⅲ. ①数据结构-高等学校-教材　Ⅳ. ①TP311. 12

中国版本图书馆 CIP 数据核字（2012）第 218338 号

出版发行 / 北京理工大学出版社
社　　址 / 北京市海淀区中关村南大街 5 号
邮　　编 / 100081
电　　话 / (010)68914775（办公室）　68944990（批销中心）　68911084（读者服务部）
网　　址 / http: // www.bitpress.com.cn
经　　销 / 全国各地新华书店
印　　刷 / 北京市通州富达印刷厂
开　　本 / 787 毫米×1092 毫米　1/16
印　　张 / 11. 5
字　　数 / 264 千字
版　　次 / 2012 年 9 月第 1 版　　2012 年 9 月第 1 次印刷　　责任编辑 / 钟　博
印　　数 / 1～4 000 册　　责任校对 / 陈玉梅
定　　价 / 26. 00 元　　责任印制 / 王美丽

前言

数据结构是计算机专业的必修课程之一，学习数据结构是为了让学生学会分析计算机加工的数据对象的特性、设计数据的组织方法，能够选择合适的数据逻辑结构和存储结构完成相应的运算（操作），把现实世界中的问题转化为计算机可识别的表示，用计算机帮助我们处理各种问题。在整个教学和学习过程中，如何分析问题、解决问题是关键，也就是说，解题能力的培养和编程技巧的训练是教学和学习的重点。为了帮助学生更好地学懂数据结构的基本原理、学会使用数据结构的方法来分析并解决问题以及掌握各种实用的编程技巧，很多学校的计算机专业在数据结构课程之后同时开设了一些实践类数据结构的相关课程，比如数据结构课程设计、数据结构实验、数据结构综合设计等。本书就是针对数据结构实践教学编写的数据结构综合设计实验教程，它在原来数据结构的基础上增加了一些实践习题的讲解和针对某种数据结构的基础实践题目，以及针对几种数据结构的综合实践题目。除此之外，本书还将原数据结构中一些较难的理论单独列出，以弥补因数据结构课时不足等问题造成的部分知识不全面。在综合设计中教师可以根据实际情况选讲其中的内容。

作者长期进行数据结构教学及程序设计类实践教学，在数据结构的教学中作者体会到，很多理论的知识讲解起来并不难，学生理解也不难，但一旦要把它们变成实践，总会有学生摸不着头脑，找不到合适的方法。理解课程内容与应用课上所学来解决实际问题之间存在着明显差距，要想理解和巩固所学的基本概念、原理和方法，掌握所学的基本知识、技能，达到融会贯通、举一反三的目的，就必须多做、多练、多见（见多识广）。这也说明了数据结构及算法设计完成的质量与基本的程序设计素质的培养是密切相关的。在程序设计类实践教学中作者也发现，很多实践是可以和数据结构结合的，但由于没有明确的说明，学生往往是只会编程，不知其然和所以然。为什么不能将二者结合，既能帮助学生应用数据结构，又能提高学生编程能力呢？于是作者致力于数据结构综合实践的研究，将各种实际问题抽象简化，用基本数据结构或者综合数据结构去实现实际问题中的数据组织、存储和操作，用简单的算法解决问题，这样既能帮助学生更好地理解数据结构、学会应用数据结构，又能在编程中使用更合理的设计方法和程序结构，增加程序的可行性、健壮性、灵活性和可移植性，提高学生编程的质量。正是为了达到上述目的，本书中通过一些实际的应用，对一些重要的数据结构和算法进行解读。经过循序渐进地训练，可以使读者掌握更多的程序设计技巧和方法，提高分析问题、解决问题的能力。

本书内容共分 8 章，前 4 章为基本数据结构的应用，第 5 章为多种数据结构的综合应用，第 6～8 章为数据结构知识扩展（部分数据结构书中含有这部分内容，但由于课时等原因讲解较少或未讲，还有部分数据结构书中没有这部分内容）。第 1 章是数据结构中链表的应用，首先简要介绍了链表的一些基本知识，然后通过“超长整数运算”和“商品库存管

理”等应用实例来巩固对链表知识的理解和使用；第 2 章为数据结构中栈和队列的应用，在介绍基本栈和队列的概念后用“表达式运算”和“停车场管理”两个应用实例来讲解如何使用栈和队列；第 3 章为数据结构中树的应用，这里除了介绍二叉树的基本概念外着重介绍了 B 树和 B+树，并且应用 B+树来解决图书管理问题；第 4 章图的应用部分介绍了图的基本概念，然后介绍了图中常用的最短路径和关键路径算法，通过公园导游问题来巩固图结构的最短路径算法的应用。第 5 章用 3 个实例将不同的数据结构综合起来应用：银行业务模拟将链表和队列综合起来，交通咨询管理将链表、文件和图结合起来共同解决问题，学生成绩管理系统使用了数组结构和文件，还用到了排序和查找算法来管理学生的信息和各科成绩。在本书最后，列出了一些实用的数据结构实践题目，可以方便老师在实践教学中选用或者学生自己选做。

本综合设计实验教程有如下特点：

（1）它在所学数据结构的基础上开设，从授课内容来讲，主要是补充数据结构课程中所学内容的不足，将数据结构中未来得及讲的内容在课程设计中变得更加明确，将图、树的应用，动态存储，外排序算法，文件等内容进行更深入的探讨，以完善数据结构的课程体系。

（2）从实践上来讲，它独立于具体的一本数据结构教科书，重点放在数据的存储以及在此存储结构上所实现的各种重要和典型的算法上，以较多的应用实例来涵盖数据结构这门课程所要求的各类重要的基础知识。

（3）它结合实际应用的要求，使课程设计既覆盖教学所要求的知识点，又接近工程的实际需要。通过实践激发学生的学习兴趣，调动学生的学习积极性，引导他们根据实际问题的需求，训练自己实际分析问题和解决问题以及编程的能力。

（4）实践内容基本上按数据结构教学的顺序设计，让学生循序渐进地学习，加深对数据结构内容的了解。另外，本书也有较大的综合课程设计，以期进一步锻炼学生的动手能力。

本书每章都选择了一两个设计实例，这些实例内容丰富、涉及面广，而且比教学实验复杂一些，涉及的更深一些，但更加实用一些，能给学习数据结构这门课程的读者以启发，达到让读者掌握相关知识和开阔视野的目的。对于各章内容中列出的实践题目，书中都进行了分析和讲解，对与之相关的算法进行了分析设计并将主要功能代码实现也相应地列出。对于书中最后的实践题目只提出了要求，希望学生能自己根据各章的内容和对数据结构的理解去独立设计实现，完成分析问题、解决问题的过程，进而学懂、学会数据结构。

本书由刘城霞、蔡英、吴燕和田英爱共同编写，其中部分实验题目的收集和整理由吴燕老师完成，还有部分学生在本书编写的过程中担任了部分程序调试的工作，在此表示深深的感谢。本书在编写的过程中得到了王铁峰老师、李宁老师、张仰森老师、陈昕老师的大力支持和帮助，在此表示衷心的感谢。另外，本书还参考了大量其他作者的书籍和资料等，在此致以诚挚的谢意。由于作者水平有限、时间仓促，书中难免存在缺点和错误，殷切希望广大读者及同行们批评指正。

作　者

目 录

基础应用篇

综合应用篇

知识扩展篇

基础应用篇

第1章 数据结构中链表的应用

§1.1 链表的基础概念

数据结构总体上可分为线性结构、树形结构和图状结构三大类。线性表是属于数据结构中的线性结构，而链表就是线性表的链式存储结构。图 1.1 表示了线性表（a_0，a_1，…，a_{n-1}）的元素间的逻辑关联关系，其中，a_{i-1}称为 a_i 的前驱，a_{i+1}称为 a_i 的后继（$0<i<n-1$）。

1. 单链表

链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的。单链表由一系列结点（链表中的每一个元素称为结点）组成，结点可以在运行时动态生成。每个结点包括两个部分：一个是存储数据元素的数据域；另一个是存储下一个结点地址的指针域。结点的结构如图 1.2 所示。

图 1.1 线性表的元素逻辑关联示意图

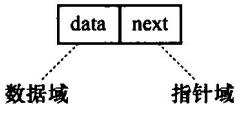

图 1.2 结点结构图

结点定义：

```
Typedef struct node{
        datatype data;
        struct node *next;
}listnode;
```

n 个结点（a_i（$0\leqslant i\leqslant n-1$）的存储映像）链接成一个链表，即线性表（$a_0$，$a_1$，…，$a_{n-1}$）的链式存储结构，如图 1.3 所示。

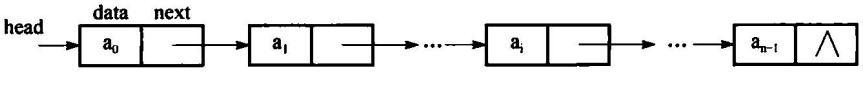

图 1.3 链表结构示意图

由于此链表的每个结点中只包含一个地址域，所以又称线性链表或单链表。

链表定义：

```
typedef listnode *linklist;
linklist head;
```

链表定义好后，最主要的操作就是插入元素和删除元素，还可以通过不断地插入元素建立这个单链表。

在单链表中第 i 个位置前插入数据元素

```
void insertnode(linklist head,datetype x,int i)
    {
        listnode* p,*q;
        p= getnode(head,i-1);              //找到第 i 个位置前的第(i-1)的位置
        if(p==NULL)
            error("position error");
        q=(listnode *)malloc(sizeof(listnode));
        q->data=x;
        q->next=p->next;
        p->next=q;
    }
```

其中，getnode（head，i－1）是找到链表中的第 i－1 个位置，它的定义为：

```
listnode *getnode(linklist head,int i) //找到以 head 为表头的链表中的第 i 个位置
 {
        int j;
        listnode *p;
        p=head;
            j=0;
        while(p->next && j<i) /*遍历第 i 个结点前的所有结点*/
          { p=p->next;
            j++;
          }
        if (i==j)
          {
              printf("%c\n",p->data);
              return p;
          }
        else
              return NULL;
 }
```

具体插入元素示意图如图 1.4 所示。

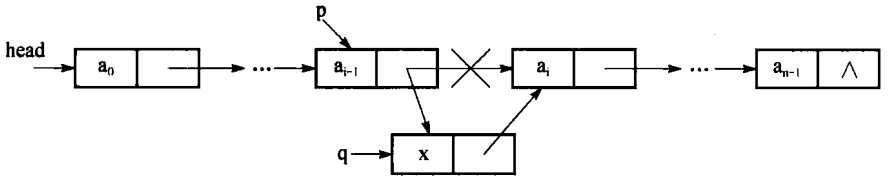

图 1.4 插入数据元素

在单链表中删除第 i 个位置的数据元素：

```
void deletelist(linklist head,int i)
    {
       listnode *p,*r;
       p=getnode(head,i-1);      //找到第 i 个位置之前的第(i-1)的位置
       if(p==NULL || p->next==NULL)
            return ERROR;
       r=p->next;
       p->next=r->next;
       free( r );
    }
```

具体删除元素示意图如图 1.5 所示。

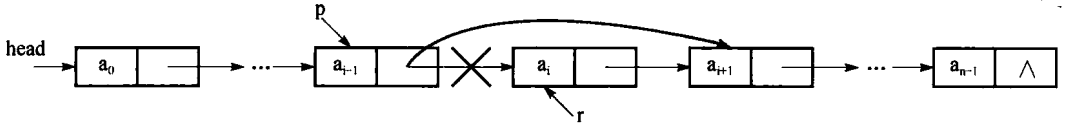

图 1.5 删除数据元素

当然对于单链表的基本操作还有很多，比如创建、取第 i 个元素值，查找某元素位置、清空链表、遍历输出所用元素等，如进行单链表设计实现时，需都一一考虑。另外，有时为操作方便，总是在链表的第一个结点之前附设一个头结点，头指针 head 指向该头结点。头结点的数据域可以不存储任何信息（或存储链表长度等信息）。

2. 循环链表

循环链表（Circular Linked List）是一种头尾相接的链表（见图 1.6）。其特点是最后一个结点的指针域指向链表的头结点，整个链表的指针域链接成一个环。从循环链表的任意一个结点出发都可以找到链表中的其他结点，使得表处理更加方便、灵活。

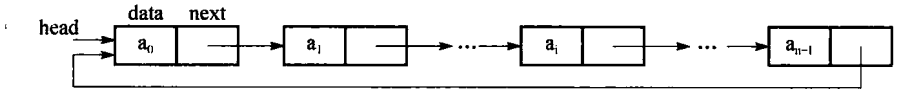

图 1.6 循环链表示意图

对于单循环链表，除链表的合并外，其他的操作与单线性链表基本上一致，仅仅需要在单线性链表操作算法基础上作以下简单修改。

(1) 判断是否是空链表：

```
head==NULL;
```

(2) 判断是否是表尾结点：

```
p->next==head;
```

3. 双向链表

双向链表是对单链表的改进。当对单链表进行操作时，有时要找到某个结点的直接前驱，由于没有指向前驱的链，必须从表头开始查找。这是因为单链表每个结点只有一个存储直接后继结点地址的链域，那么能不能定义一个既有存储直接后继结点地址的链域，又有存储直接前驱结点地址的链域的这样一个双链域结点结构呢？因此双向链表就诞生了。在双向链表中，结点除含有数据域外，还有两个链域：一个存储直接后继结点地址，一般称之为右链域；另一个存储直接前驱结点地址，一般称之为左链域。具体的结点结构图如图 1.7 所示。

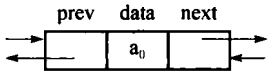

图 1.7　双链表结点结构图

双链表结点的定义：

```
typedef struct dlistnode{
        datatype data;
        struc dlistnode *prior,*next;
    }dlistnode;
typedef dlistnode *dlinklist;
```

同样，双向链表的插入操作为：

```
void dinsertbefor(dlistnode *p,datatype x)      //将 x 插入 p 结点后
      {
        dlistnode *q=malloc(sizeof(dlistnode));
        q->data=x;
        q->prior=p->prior;
        q->next=p;
        p->prior->next=q;
        p->prior=q;
      }
```

双链表插入数据元素示意图如图 1.8 所示。

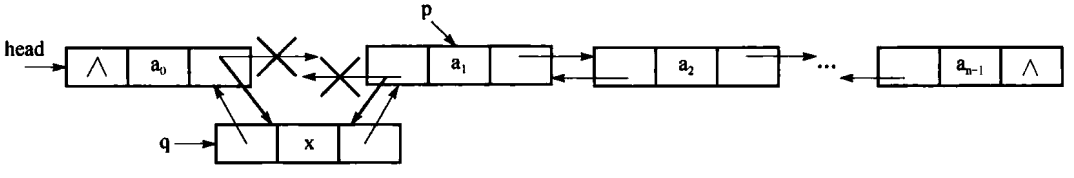

图 1.8　双链表插入数据元素

双向链表的删除操作为：

```
void ddeletenode(dlistnode *p)      //将 p 结点删除
    {
```

```
        p->prior->next=p->next;
        p->next->prior=p->prior;
        free(p);
    }
```

双链表删除数据元素示意图如图 1.9 所示。

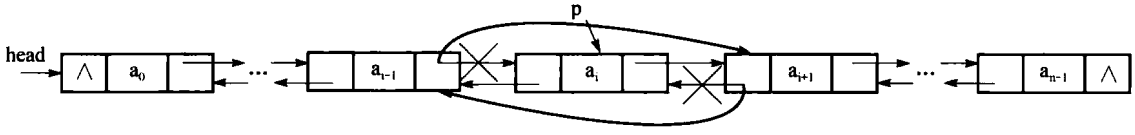

图 1.9 双链表删除数据元素

双链表比单链表更加复杂些，对于双链表的其他基本操作，比如创建、查找元素位置、取元素、清空等操作也不再一一详述，学生可以自行实现。

将双链表的头尾相接，可以得到双向循环链表，其基本结构如图 1.10 所示。

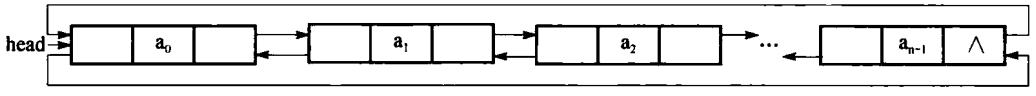

图 1.10 双向循环链表示意图

下面看一个利用双向链表（或双向循环链表）的应用实例，来进行超长整数的加减法运算。

§1.2 超长整数的加减法运算

1. 需求分析

在计算机中有存储整数的数据类型，比如整型、长整型、无符号整型等，但它们的存储位数是有限的，这与具体的语言、具体的编译环境有关。一般 C 语言中，整型为 16 位长，长整型为 32 位长。但是在一些应用中要求的数据非常大，比如在公钥密码学中的模数要求是一个大素数，这个数如果太小会影响密码的安全性。当然，对于这些大数要进行运算，因此，有必要考虑一些超长整数的运算。

这里要求设计一个实现任意长的整数进行加减法运算的算法程序。具体的是要求设计一个存储大整数的数据结构及其运算的算法，能进行任意位数的整数加减法运算，这其中任意长整数可为正数，也可为负数，数据可以由键盘输入或者是由数据文件中读取。

2. 方案设计

为了方便存储数据，可采用长度可以任意伸缩的链表来存储数据。又由于运算通常是从低位到高位进行，而输入数据通常是由高位到低位输入，那么存储数据的链表可以双方向进行操作，所以采用利用双向链表实现长整数的存储，每个结点可以含一个整形变量，也可以含多个整型变量，这里为了方便运算，采用一个结点内存储一个整型变量的策略。

进一步讲，如果从链表尾部直接需要找到链表头部或者从链表头部可以方便地找到链表尾部，可以将链表首尾相接，形成双向循环链表［如果不想采用双向循环链表，也可以对头

和尾分别设置指针（head，tail）指示，需要用到表头的时候直接通过 head 即可找到，需要用到表尾的时候通过 tail 就可以找到]。输入数据及结果形式可以按中国对于长整数的表示习惯，每四位一组，组间用逗号隔开。也可以按照国际惯例，每 3 位一组进行输入输出，组件用逗号隔开。这里采用第一种方案进行输入输出。

数据结构的结点设计图如图 1.11 所示；双向链表循环链表设计图如图 1.12 所示。

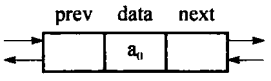

图 1.11　结点设计图

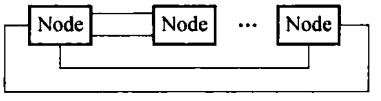

图 1.12　双向循环链表设计图

```
typedef struct DuLNode
{
    ElemType data;
    struct DuLNode *prior,*next;
}DuLNode,*DuLinkList;
```

该双向循环链表可以进行的基本操作包括：

① 构造一个空的双向循环线性表，L 为头结点：

Status InitList_DuL (DuLinkList &L);

② 在 L 中确定第 i 个元素的位置指针 p：

DuLinkList GetElemP_DuL (DuLinkList L, int i);

③ 在双链循环线性表 L 中第 i 个位置之前插入元素 e，i 的合法值为 1<=i<=List Length_DuL(L)+1：

Status ListInsert_DuL (DuLinkList &L, int i, ElemType e);

④ 删除带头结点的双链循环线性表 L 中第 i 个元素，i 的合法值为 1<=i<=List Length_DuL (L)：

Status ListDelete_DuL (DuLinkList &L, int i, ElemType &e);

⑤ 若 L 为空表，则返回 TRUE；否则返回 FALSE：

bool ListEmpty_DuL (DuLinkList L);

⑥ 线性表 L 已存在，返回 L 中数据元素个数：

Status ListLength_DuL (DuLinkList L);

⑦ 若线性表 L 已存在，将 L 置为空表；销毁线性表 L：

Status ClearList_DuL (DuLinkList &L);

Status DestroyList_DuL (DuLinkList &L);

⑧ 用 e 返回 L 中第 i 个数据元素的值；取出倒数第 i 个位置的数据元素：

void GetElem_DuL (DuLinkList L, int i, ElemType &e);

void GetLastElem_DuL (DuLinkList L, int i, ElemType &e);

⑨ 输出双向循环链表 L 中的数据：

Void PrintList_DuL (DuLinkList L);

⑩ 返回第 n 个 e 的在链表中位置 n 的合法值为 0<=n<=Counter_DuL(L,n)：

int Position_DuL (DuLinkList L, int n, ElemType e)；

我们用键盘输入的任意长数字字符串，以“#”作为输入长整数数据的结束符，进行数据的输入。输入的字符串在内部用双向循环链表来接收并存储，具体输入数据的处理过程如图 1.13 和图 1.14 所示。

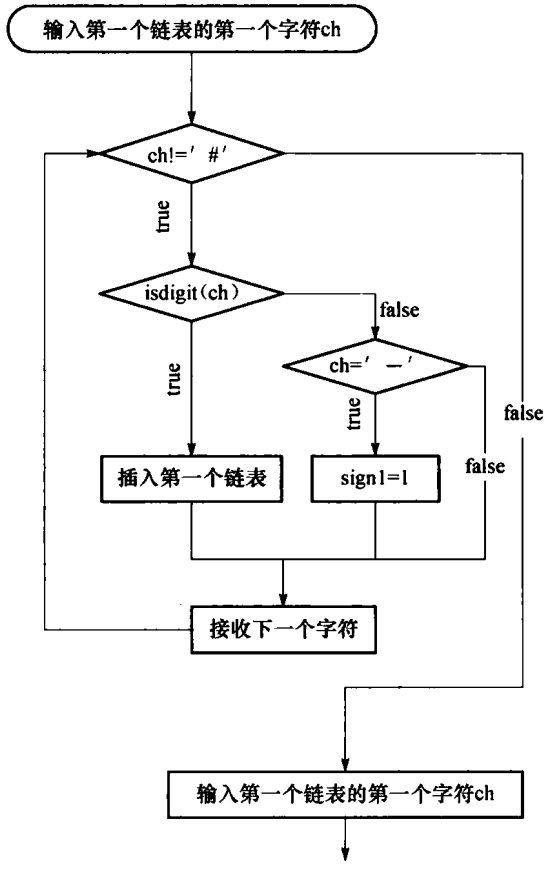

图 1.13 创建第一个长整数链表

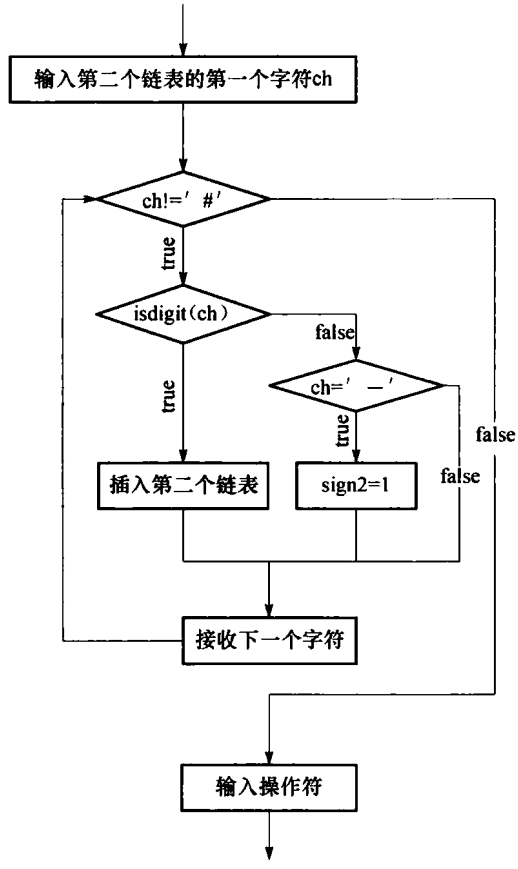

图 1.14 创建第二个长整数链表

其中 sign1 和 sign2 表示的是数的符号位，正数为 0，负数为 1。把数据存储进双向链表后，根据输入的运算符号判断是加法还是减法，再结合输入数据的正负符号进行真正的加法或减法的运算。表达式的形式有八种可能：a+b，a−b，a+(−b)，a−(−b)，−a+b，−a−b，−a+(−b)，−a−(−b)（a，b 均为正数）。它们可归结为四种：a+b，a−b，−a+b，−a−b（a，b 均为正数）

最后本质上为：a+b，a−b（a，b 均为正数）。两个长整数的运算如图 1.15 所示。其中，结合运算和数的符号确定运算是加法后可以直接进行数据的加法运算，但如果运算是减法运算，则还需要检查减数和被减数的大小；如果是较大数减较小数的形式，则直接进行减法；如果是较小数减较大数的形式，则改变结果的符号位，将它转换为较大数减较小数。

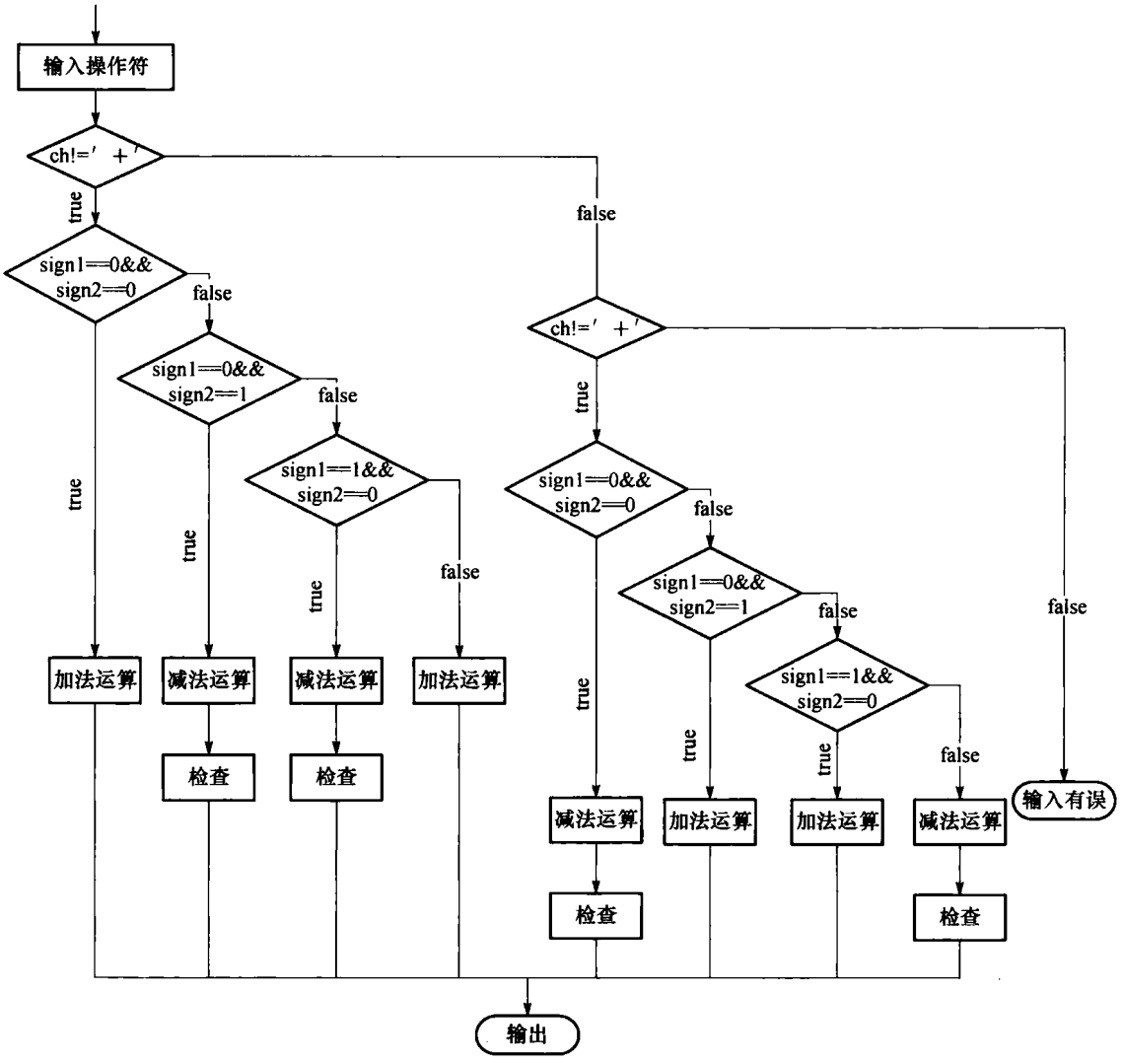

图 1.15　两个长整数进行运算

具体加法运算的过程和减法运算的过程如图 1.16 和图 1.17 所示。

其中，first 为取出的第一个运算数中一个结点的内容；second 为取出的第二个运算数中一个结点的内容；carry 为加法时用来保存进位的变量；borrow 为减法时用来保存借位的变量；result 为结果链表。具体的加减法算法为

```
void plus (DuLinkList firstNum,DuLinkList secondNum,char reusltflag)
{
 DuLinkList result;
 ElemType first,second;
 ElemType templResult;
 int firstCount=ListLength_DuL(firstNum);      //获取第一个运算数的结点个数
 int secondCount=ListLength_DuL(secondNum);    //获取第二个运算数的结点个数
 int maxCount=firstCount;
```