算法竞赛宝典

第三部
基础数据结构

张新华 编著

3

清华大学出版社

算法竞赛宝典

张新华 编著

清华大学出版社
北 京

图书在版编目(CIP)数据

基础数据结构/张新华编著. --北京：清华大学出版社，2016
(算法竞赛宝典)
ISBN 978-7-302-40950-2

Ⅰ. ①基… Ⅱ. ①张… Ⅲ. ①程序设计－数据结构－数学参考资料 Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字(2015)第 166259 号

责任编辑：纪海虹　刘美玉
封面设计：张新华
责任校对：王荣静
责任印制：杨　艳

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京密云胶印厂
经　　销：全国新华书店
开　　本：185mm×260mm　　印　张：17.75　　字　数：427 千字
版　　次：2016 年 4 月第 1 版　　印　次：2016 年 4 月第1次印刷
印　　数：1～4000
定　　价：42.00 元

产品编号：062996-01

FOREWORD

<<< 前言

为什么编写这套书

随着计算机逐步深入人类生活的各个方面，利用计算机及其程序设计来分析、解决问题的算法在计算机科学乃至整个科学界的作用日益明显。相应地，各类以算法为主的程序设计竞赛也层出不穷：在国内，有全国青少年信息学奥林匹克联赛（National Olympiad in Informatics in Provinces，NOIP），该联赛与全国中学生生物学联赛、全国中学生物理竞赛、全国高中数学联赛、全国高中学生化学竞赛，并称为国内影响力最大的“五大奥赛”；在国际，有中学生的国际信息学奥林匹克竞赛（International Olympiad in Informatics，IOI）、面向亚太地区在校中学生的信息学科竞赛（即亚洲与太平洋地区信息学奥林匹克，Asia-Pacific Informatics Olympiad，APIO）和国际大学生程序设计竞赛（ACM International Collegiate Programming Contest，ACM/ICPC）等。

各类算法竞赛要求参赛选手不仅必须具有深厚的计算机算法功底、快速、准确的编程能力以及创造性的思维，而且还必须具备团队合作精神和抗压工作的能力，因此算法竞赛在高校、IT 公司和其他社会各界中获得越来越多的认同和重视。算法竞赛的优胜者，更是微软、Google、百度、Facebook 等全球著名 IT 公司争相高薪招募的对象。因此，不仅是各类参加算法竞赛的选手，即使是不参加此类竞赛的很多研究工作者和从事 IT 行业的人士，都希望能获得这方面的专业训练并从中得到一定的收获。

但是，长期以来，算法竞赛的内容以其高难度、高要求而困住了许多有志于参加算法竞赛的爱好者的求索脚步。部分已出版的同类书籍或艰深难懂、或教条灌输，使不少爱好者望而却步，半途而废，以至于研究算法成了一种“曲高和寡”的学问，严重制约了算法竞赛优秀人才的培养。

殊不知，书上多数看似复杂难懂的知识或内容，其实总是可以找到一种深入浅出、化难为易的表述方法，使读者能看得懂、学得透，并最终达到触类旁通、举一反三的学习目标。因此本套丛书的写作初衷，是试图将看似晦涩难懂的算法表述得轻松、活泼、易懂。同时也是为了弥补国内算法竞赛以及普及读物方面的不足。

希望本套丛书的出版，能够给那些学有余力的中学生、计算机专业的大学生、程序算法爱好者以及 IT 从业者提供一个学习计算机科学的机会。

为什么要学习算法

经常有人说：我不学算法也照样可以编程写软件。那么，为什么还要学习算法呢？

首先，算法（algorithm）一词源于算术（algorism），具体地说，算术方法是一个由已知推

求未知的运算过程。后来，人们把它推广到一般，即把进行某一工作的方法和步骤称为算法。一个程序要完成一个任务，其背后肯定要涉及算法的实现，算法的优劣，直接决定了程序的优劣，因此，算法是程序的灵魂。学好了算法，就能够设计出更加有效的软件，以最为有效的方式完成复杂的功能。例如，设计完成一个具有较强人工智能的人机对弈棋类游戏，程序员没有深厚的算法功底是根本不可能实现的。

其次，算法是对事物本质的数学抽象，是初等数学、高等数学、线性代数、计算几何、离散数学、概率论、数理统计和计算方法等知识的具体运用。真正学通计算机的人（不只是“编程匠”）都对数学有相当的造诣，既能用科学家的严谨思维来求证，也能用工程师的务实手段来解决问题——而这种思维和手段的最佳演绎就是“算法”。学习算法，能锻炼我们的思维，使思维变得更清晰、更有逻辑，变得更有深度和广度。学习算法更是培养逻辑推理能力的非常好的载体之一。因此，学会算法思想，其意义不仅仅在算法本身，更重要的是，对日后的学习生活和思维方式也会产生深远的影响。

最后，学习算法本身很有意思、很有趣味。所谓“技术做到极致就是艺术”，当一个人真的沉浸到算法研究中时，他会感受到一个个精妙绝伦的算法的艺术之美，他会为它惊人的运行速度和巧夺天工的构思而深深震撼，并从中体会到一种不可言喻的美感和愉悦。当然，算法的那份优雅与精巧虽然吸引人，却也令很多人望而生畏。事实证明，对很多人来说，学习算法是一件很痛苦的事情。

本套丛书的特色

本套丛书的第一个特点是具备一定理解能力的读者可以凭自学看懂书中的多数内容。由于地区的不平衡性等原因，很多算法竞赛选手及算法爱好者面对的一个最大难题是没有合适的教材以及找不到合适的导师给予引导。考虑到这种情况，本书作者尽可能地对书中每一个难点和重点都进行了深入仔细的讲解，此外，书中的绝大多数题目均附有带注释的完整源代码及测试数据供读者参考和检测。当然，不管怎样，你永远也不要指望阅读一本算法书能像阅读一本通俗小说那般轻松惬意。

本套丛书的第二个特点是摒弃了一些算法书过于“学究气”，片面强调学术性、严谨性而忽略了可读性的“四平八稳”的表述手法，代之以轻松、活泼的故事情节和穿插以历史、文化、科技等要素的情境导入，读者不仅在学习过程中获得编程的乐趣，还能够跟随书中的众多人物，一同经历一场奇妙的冒险之旅。本书作者热切希望读者不仅最终能够在计算机科学的领域大展宏图，更能够在故事中获得感悟，对社会、人生、理想、价值和信念等有更深刻的思考和认识。

本套丛书的第三个特点是绝大多数题目均采用了“多向思考”“一题多解”和“一题多变”的解决方法，其目的主要有三点：一是为了充分调动读者思维的积极性，提高综合运用已学知识解答问题的技能技巧；二是为了锻炼读者思维的灵活性，促进知识和智慧的增长；三是训练思维深度和广度，引导读者灵活地掌握知识的纵横联系，培养和发挥读者的创造性。所以，这种训练，决不能简单地看作是编程技巧的花哨卖弄，而是通过这种训练，培养读者思维的敏捷性，促进读者智力和思维的发展，提高读者的变通能力与综合运用所学知识的能力。读者若能坚持这种思维训练方法，必能起到意想不到的良好学习效果。

本套丛书内容简介

本套丛书的第一部——《语言及算法入门》，主要介绍在算法竞赛中需要用到的 C++语

言的语法知识及一些简单算法的运用。但与一般C++语言入门书不同的是，本部书在介绍C++语言的同时，更加侧重于数学思维的培养和简单算法的应用，因此其学习难度远高于一般市面上的C++语言入门书。书中很多表面看上去似乎非常简单的题目，由于采取了“一题多解”及“数学求解”等方法，其程序复杂度直线上升。因此这就要求读者具备一定的数学功底和思维能力，并且需要花费相当长的时间去思考和练习，才可能深刻理解题目的本质和内涵。此外，为了防止初学者原封不动地照抄源代码而不理解代码真实含义的“恶习”，书中凡可写可不写源代码的地方，一律采用伪代码形式表述，当然，在出版社网站上，读者可以下载到书中所有的源代码及测试数据等资源。

本套丛书的第二部——《基础算法艺术》，重点介绍了各种基础算法，如分治算法、贪心算法、枚举算法、动态规划算法等。书中的绝大多数题目都采用了“多向思考”、“一题多解”和“一题多变”的方式来解决。读者不仅可以通过出版社网站下载的简单测试数据验证所写程序的正确性，还可以根据书中标注的题目原始出处，访问相关的在线评测网站提交所写代码进行测试。需要注意的是，与多数教材从头读到尾的习惯不同的是，本书的各章节划分并不是严格按从难到易、从简到繁的顺序编排，各章节基本都可以独立阅读，互不影响。读者在阅读过程中，有时可根据需要翻到其他章节学习完相关内容后再返回本章节继续学习。此外，每一道题的各种算法按难易程度粗略地以★号表示，★号越多，表示难度越大，读者若遇到难度过大的题，不必硬“啃”，完全可以在学习完其他章节的相关内容后再来学习。

本套丛书的第三部——《基础数据结构》，详细介绍了链表、堆栈、队列、树、图等基础数据结构的相关知识。为了便于读者的理解，本书对数据结构众多知识点进行了详细的解释和分析，随书还配有难易适中的练习题。本书中的多数题目未配置相应测试数据，读者编写的代码正确与否，需要去相关的在线评测网站提交代码进行测试。这样做是培养读者善于应用无限网络资源的能力，使读者能逐渐脱离书本的束缚，最终达到独立、自主学习的目的。

全套书中以调侃化的形式出现的公司和人名等，均为故事情节而服务，与现实无更多关联。

适合阅读本套丛书的读者

本套丛书可作为全国青少年信息学奥林匹克联赛(NOIP)系列的复赛教材及ACM国际大学生程序设计竞赛(ACM/ICPC)的参考和学习用书，还可作为计算机系学生、IT工程师、科研人员、算法爱好者的参考和学习用书。

致谢

感谢原乌鲁木齐高级中学(前身为乌鲁木齐市第六中学)的李念、刘悠然、朱生龙、陆智、李智杰、刘毅东、吕亚伟、徐兆鹏、戴文轩、杜同心、胡亚欣、徐菲鹏、戴维、张欢、郑斐、彭胡杰、肖阳等同学；感谢浙江省瑞安中学钱瑞源、潘伟达、张元煌、周望威、林展、杨丰、张颖洁、卢众意、沈小洲、肖煜伟、郑立言、金泽豪、章奎亮、蔡子豪、陈云帆、方佳豪、陈士俊、李心怡、孙一言、蔡凡、黄海同、王络、肖瑞宇、潘洲阳、陈可、张夏悦、陈振哲、林江浩、魏书扬等同学，他们在笔者的指导下，用简洁易懂的程序代码实现了本书的部分算法。

感谢全国各省市中学、大学的信息学奥赛指导老师们，他们给本套丛书提了许多真诚而有益的建议，并对本人在写书过程中遇到的一些困惑和问题给予了热心的解答。

感谢浙江省瑞安中学为本人搭建的学习、研究、竞赛的平台以及对本人工作的大力支持。

本套丛书在完成过程中，使用了 NOIP 的部分原题、在线评测网站的部分题目，并参考和收集了其他创作者发表在互联网、杂志等媒体上的相关资料，无法一一列举，在此一并表示衷心感谢。

另外，感谢我的女儿，你是我完成这套书的动力，因为这套书专为你而写。

最后要说的话

本套丛书花费了笔者巨大的精力和时间，这些年来的多数闲暇时间，都用在了这套丛书的编写和反复校验上。但尽管如此，由于时间和水平所限，书中难免存在不妥和错误之处，欢迎同仁或读者赐正，如果在阅读中发现任何问题，请发送电子邮件到 hiapollo@sohu.com 告诉本人，更希望读者对本书提出建设性意见，以便修订再版时改进。

针对本套书的题库网站正在不断完善中，网址：www.razxhoi.com。

张新华

2014 年 8 月于浙江瑞安

CONTENTS

<<< 目录

第一章

链　表

何谓链表

我们知道，一般用数组存放一组数据时，必须事先定义固定的长度(即元素个数)。这在某些问题的解决中，并不是特别的适用。例如，记录不同班级的学生数据时，由于各班人数不同，会出现开辟过大的数组导致内存浪费、开辟过小的数组导致数组元素不够用的情况。而链表可以根据需要动态开辟内存单元，是一种常见的重要数据结构。图 1.1 所示为最简单的一种链表结构。

图　1.1

链表如同铁链一样，一环扣一环，中间是不能断开的。打个通俗的比方：幼儿园老师带领小朋友出来散步，老师牵着第一个小朋友的手，第一个小朋友的手牵着第二个小朋友的手……这就是一个“链”，最后一个小朋友的手是空的。

老师即“头指针”变量，图 1.1 中以 Head 表示，它存放一个地址。链表中每一个元素称为“结点”，每个结点都应该包括两部分：一为实际元素值，一为下一结点的地址。

最后一个元素不指向其他元素，它被称为“表尾”，以“NULL”表示，“NULL”在 C++语言里指向“空地址”。

很显然，这种链表的数据结构，必须要用指针变量才能实现。

简单静态链表

下面的代码是一个简单的静态链表，它由 3 个学生的数据(学号，成绩)的结点组成。请考虑：(1)head 的作用？(2)p 的作用？

```
1  //简单静态链表
2  #include <iostream>
3  using namespace std;
4
5  struct  student
6  {
7    long num;
8    float score;
9    struct student *next;                    //该指针指向 student 类型的结构体
10 };                                         //注意必须有分号
11
12 int main()
13 {
14   struct student a,b,c, *head, *p;
15   a.num=34341;   a.score=81.5;            //赋值
16   b.num=34343;   b.score=97;
17   c.num=34344;   c.score=82;
18   head=&a;                                 //将 a 的地址给 head
19   a.next=&b;
20   b.next=&c;
21   c.next=NULL;
22   p=head;
23   do                                       //输出记录
24   {
25       cout<<p->num<<" "<<p->score<<endl;
26       p=p->next;
27   }while(p!=NULL);
28   getchar();
29 }
```

处理动态链表的函数

1. malloc

函数原型为

```
void *malloc(unsigned int size);
```

作用是在内存的动态存储区中分配一个长度为 size 的连续空间。此函数返回的是一个指向分配域起始地址的指针，如果此函数未能成功地执行(如内存空间不足)，则返回空指针(NULL)。

2. calloc

函数原型为

```
void *calloc(unsigned n,unsigned size);
```

作用是在内存的动态存储区中分配 n 个长度为 size 的连续空间。函数返回一个指向分配域起始地址的指针；如果分配不成功，则返回 NULL。

3. free()

函数原型为

```
void free(void * p);
```

作用是释放由 p 指向的内存区，使这部分内存区能被其他变量使用。

动态链表的准备工作

一个完善的动态链表程序应该具有以下基本功能：建立链表、插入结点、删除结点、打印链表、释放链表等。扩展的动态链表程序还可能有获得链表长度、获得当前结点、查找结点位置、连接两个链表、比较两个链表等功能。下面将逐个实现其功能代码。

为了程序的易读性和可扩展性，有时需要在程序开头先进行预定义处理。请务必领会下面代码的用意，否则将影响对以后代码的理解。

```
#include <stdlib.h>
#include <stdio.h>
typedef int ElemType;                    //以 ElemType 代表 int 型数据
typedef struct List *link;               //以 link 代表链表指针
typedef struct List Lnode;               //以 Lnode 代表链表结点

struct List
{
   ElemType data;                        //此处仅以一个整型变量为例
   struct List *next;
};
```

主函数的建立：下面的主函数只是一个简单调用各功能子函数的示范例子，读者可自行修改和添加代码以完成更复杂的任务。请根据主函数的代码考虑各功能子函数的原型应如何建立。

```
int main()
{
   int l;link head1;link head2;
   head1=create(head1);                                  //建立链表 1
   head2=create(head2);                                  //建立链表 2
   connect(head1,head2);                                 //连接两个链表
   head1=insert(head1,888,5);                            //在位置为 5 处插入元素 888
   head1=del(head1,3);                                   //删除一个结点
   display(head1);                                       //打印链表
   printf("\n  lenth is %d\n", lenth(head1));            //打印链表长度
   printf("\n  get is %d\n", get(head1,3));              //获得当前结点值
   printf("\n  locate 12 is %d", locate(head1,12));      //查找元素 12 所在的位置
   head=setnull(head);                                   //释放链表
}
```

链表的建立

由主函数调用 create()函数的方式可知，该函数应该返回一个结点的指针，输入的参数也应该是一个结点指针。参考代码如下。

```
1  link create(link Head)
2  {
3    ElemType newData;
4    link NewPoint;
5    /*先建立一个结点*/
6    Head=(link)malloc(sizeof(Lnode));
7    printf("please input number: \n");
8    scanf("%d",&newData);
9    Head->data=newData;                    //结点赋值
10   Head->next=NULL;                       //结点指向空地址
11
12   while(1)                               //继续添加结点
13   {
14     NewPoint=(link)malloc(sizeof(Lnode));  //开辟一个结点空间
15     if(NewPoint==NULL)                   //如果开辟空间失败,则返回
16       break;                             //此判断语句在某些类型的竞赛中用处不大,可忽略
17     printf("please input number: input '-1' means exit\n");
18     scanf("%d",&newData);
19     if (newData==-1)                     //输入-1则添加结点结束并返回 head
20       return Head;
21     NewPoint->data=newData;
22     NewPoint->next=Head;
23     Head=NewPoint;
24   }
25   return Head;
26 }
```

链表的显示

该子函数无返回值，输入参数为链表的头指针，请考虑：指针 p 的作用是什么？如果不用指针 p，直接用 Head 这个指针会出现什么后果？

```
1  void display(link Head)
2  {
3    link p;p=Head;
4    if(p==NULL)
5      printf("\nList is empty\n");
6    else
7      do
8      {
9        printf("%d ",p->data);
```

```
10          p=p->next;
11      }while(p!=NULL);
12  }
```

结点的插入

```
1   link insert(link Head,ElemType x,int i)
2   {
3     link NewPoint,p=Head;
4     int j=1;
5     NewPoint=(link)malloc(sizeof(Lnode));
6     NewPoint->data=x;
7     if(i==1)      //如果插入位置为第一个结点时
8     {   NewPoint->next=Head;   Head=NewPoint; }
9     else
10    {
11      while(j<i-1 && p->next!=NULL)
12      {      p=p->next;    j++;   }
13      if(j==i-1)
14      {
15        NewPoint->next=p->next;
16        p->next=NewPoint;
17      }
18      else   printf("insert is Failure,i is not right!");
19    }
20    return Head;
21  }
```

结点的删除

例如下面这个链表：

(A)→(B)→(C)→(D)→(E)→…→NULL

因为链表中唯一能够找到元素的办法是通过它上一个元素的指针，所以如果我们将某个元素直接删除，这个元素所指向的元素和它之后的所有元素都没有办法再找到了，为了解决这个问题，一般采取这样的办法：记录下要删除的元素之前的那个元素，在删除元素之前，把这个元素的指针指向要删除的那个元素指针指向的元素，比如说现在要删除这个链表的元素B，先找到它之前的元素 A，在删除 B 之前将 A 的指针指向 C，然后再删除 B，这样在删除 B 之前，就已经把 B 从链表里面剔了出来，如图 1.2 所示。

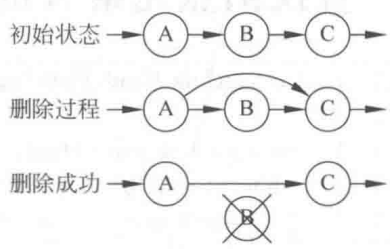

图 1.2

```
1   link del(link Head,int i)
2   {
```

```
3    int j=1; link p,t; p=Head;
4    if(i==1)
5    {   p=p->next;   free(Head);   Head=p; }
6    else
7    {
8       while(j<i-1 && p->next !=NULL)
9       {  p=p->next;       j++;     }
10      if(p->next!=NULL && j==i-1)
11        {  t=p->next;       p->next=t->next;     }
12      if(t!=NULL)  free(t);
13   }
14   return Head;
15 }
```

上面的例子中操作符 free 用来删除一个变量，即在内存中释放某个变量所对应的地址，使得之后系统可以利用这块内存做别的事情，这样比较节省内存空间。我们应养成及时释放无用的内存空间的好习惯。

获得结点元素值

```
1  ElemType get(link Head,int i)
2  {
3     int j=1;
4     link p; p=Head;
5     while(j<i && p!=NULL)
6     {     p=p->next;     j++;  }
7     if(p!=NULL)
8          return(p->data);
9     else
10         printf("data is error!");
11         return -1;
12 }
```

查找结点元素 X 的位置

```
1  int locate(link Head,ElemType x)
2  {
3     int n=0;link p;p=Head;
4     while(p!=NULL && p->data !=x)
5     {  p=p->next;      n++;  }
6     if(p==NULL)
7          return -1;
8     else
9          return n+1;
10 }
```