ACM程序设计培训教程

吴 昊 主编　蒋 斌 廖 波 朱宁波 参编

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

ACM 程序设计培训教程

吴 昊 主编

蒋 斌 廖 波 朱宁波 参编

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书针对ACM程序设计竞赛出现比较多的16个方面的问题，通过案例的方式说明解决问题的方法。由于数据结构使用非常多，对不属于16个专门问题的知识我们也进行了介绍。

本书不是这些专门问题的教科书，所以对这些问题所涉及知识的介绍不多，主要是分析一个个案例，介绍专属于ACM程序设计的方法和技巧。

图书在版编目（CIP）数据

ACM程序设计培训教程/吴昊主编. —北京：中国铁道出版社，2007.8
ISBN 978-7-113-07651-1

Ⅰ.A... Ⅱ.吴... Ⅲ.程序设计—技术培训—教材
Ⅳ.TP311

中国版本图书馆CIP数据核字（2007）第122212号

书　　名：ACM程序设计培训教程
作　　者：吴　昊　蒋　斌　廖　波　朱宁波
出版发行：中国铁道出版社(100054,北京市宣武区右安门西街8号)
策划编辑：严晓舟　秦绪好
责任编辑：杨　勇　包　宁
封面设计：高　洋
封面制作：白　雪
印　　刷：北京市彩桥印刷有限责任公司
开　　本：787×1092　1/16　印张：17.5　字数：409千
版　　本：2007年8月第1版　　2007年8月第1次印刷
印　　数：1～5 000册
书　　号：ISBN 978-7-113-07651-1/TP·2224
定　　价：28.00元

序　言

当今时代，计算机教育已经成为大学教育的一个极其重要的组成部分。大学生运用计算机来充分展示自己分析问题和解决问题的能力，是对参赛选手的创造力、团队合作精神以及他们在软件程序开发过程中的创新意识的考验，同时也是检测选手们在压力下进行开发活动的能力。可以说，大学生程序设计竞赛是参赛选手展示计算机才华的广阔舞台，是大学计算机教育成果的直接体现，是信息企业与世界顶尖计算机人才对话的最好机会。

大学生程序设计竞赛是对计算机基础知识很好的检验。在程序设计大赛中，几乎所有计算机相关的基础知识（如离散数学、数据结构、计算机算法、计算理论、编译原理、数值分析、程序设计、高等数学、计算机信息安全、密码学、概率论）都得到了应用，而且涉及到计算机科学最前沿的发展。同时，大学生程序设计竞赛可以充分锻炼参赛选手综合应用所学的知识，提高分析问题和解决问题的能力，从而促进学风建设和培养全面发展优秀人才。

大学生程序设计竞赛不仅仅是对计算机学科基础知识的应用，而且是高等教育的基础学科的综合应用，它涉及数学、物理、生物、化学、商业、医学、英语等学科知识。可以这样说，大学生程序设计竞赛对于高校来讲是可以与数学、物理相媲美的基础学科。所以各所大学以及各省一向十分重视大学生程序设计竞赛在校园内的开展，将其作为学校迈向世界一流大学建设过程中的重点学生科技竞赛活动来组织实施，取得了非常好的成绩和效果。目前，全国有很多省和地区已经组织或预备组织此类竞赛，并通过竞赛活动积累了较丰富的经验，如广东省、上海市、北京市、湖南省等。

现在我国每年有 1 000 多个高校的 3 000 多支参赛队伍，参加各种级别的程序设计大赛，而且吸引着更多的程序设计爱好者在这片天地展示着自己的才华。但由于计算机发展水平的不均衡，很多刚开展程序设计竞赛的学校和大学生，难以找到一本较合适的培训教材，不能很快地提高计算机程序设计水平。

湖南大学计算机与通信学院程序设计训练队的教练组，通过几年来对 ACM 程序设计竞赛赛题的研究，积累了一些教学经验。现在将其编写为 ACM 培训教材，以填补这方面的空白。

李仁发

2007 年 5 月 30 日

前　言

ACM 国际大学生程序设计竞赛（ACM International Collegiate Programming Contest，ACM/ICPC）是由国际计算机界历史悠久、颇具权威性的组织 ACM 学会（Association for Computing Machinery，美国计算机协会）主办，是世界上公认的规模最大、水平最高的国际大学生程序设计竞赛，其目的旨在使大学生运用计算机来充分展示自己分析问题和解决问题的能力。该项竞赛从 1970 年举办至今已历 27 届，因历届竞赛都荟萃了世界各大洲的精英，云集了计算机界的“希望之星”，而受到国际各知名大学的重视，并受到全世界各著名计算机公司的高度关注，成为世界各国大学生最具影响力的国际级计算机类的赛事。此项赛事的主办目的不单是培养参赛选手的创造力、团队合作精神以及他们在软件程序开发过程中的创新意识，同时也是检测选手们在压力下进行开发活动的能力。可以说，ACM 国际大学生程序设计竞赛是参赛选手展示计算机才华的广阔舞台，是著名大学计算机教育成果的直接体现，是信息企业与世界顶尖计算机人才对话的最好机会。

当今时代，计算机教育已经成为大学教育的一个极其重要的组成部分。大学生程序设计竞赛是对计算机基础知识的检验。在程序设计大赛中，几乎所有计算机相关的基础知识（如离散数学、数据结构、计算机算法、计算理论、编译原理、数值分析、程序设计、高等数学、计算机信息安全、密码学、概率论等）都得到应用，而且涉及到计算机科学最前沿的发展。可以充分锻炼参赛者综合应用所学的知识，提高分析问题和解决问题的能力，从而促进学风建设和培养全面发展的优秀人才。

因此，现在 ACM 程序设计竞赛的影响越来越大，参赛队伍越来越多。ACM/ICPC 亚洲区竞赛总主席黄金雄先生说：“参加决赛的队伍都是程序设计的精英。”

湖南大学一直大力支持着 ACM 程序设计的工作，举办了程序设计的月赛和每年度的湖南大学校赛，建立了学校的竞赛网站 http://acm.hnu.cn，参加了各级比赛，取得了一些成绩。

为了提高学生的程序设计水平，从 2004 年开始组织程序设计的暑期培训班，从 2006 年开始，开设了全校的选修课——ACM 程序设计。以专题讲座的形式介绍 ACM 程序设计竞赛使用到的知识。这本书源自培训和上课使用的教案。

通过研究，我们发现在 ACM 程序设计竞赛中出现较多的有 16 个方面的问题，所以本书针对这 16 个问题，通过案例的方式说明解决问题的方法。由于数据结构使用非常多，对不属于 16 个专门问题的知识我们也进行了介绍。

本书不是这些专门问题的教科书，所以对这些问题所涉及知识的介绍不多，主要是分析一个个案例，介绍专属于 ACM 程序设计的方法和技巧。

由于编者水平有限，疏漏和不足之处在所难免，望读者指正为幸。

感谢湖南大学计算机与通信学院院长李仁发教授为本书作的序言，同时感谢骆嘉伟副院长给本书提供了很多很好的建议，感谢湖南大学 ACM 程序设计队队员提供的不少的实际案例。

编　者

2007 年 7 月

目　录

第1章 经典数据结构与算法

经典数据结构是所有问题的基础，牢固掌握这些基本的数据结构和算法是一项基本的要求，同时也是进一步提高自身解题技巧的必经途径。高级数据结构和算法是在这些经典数据结构与算法的基础上进行组合和灵活运用的结果。

1.1 线 性 表

线性表（Linear List）是由 n（$n\geqslant 0$）个类型相同的数据元素 a_1，a_2，…，a_n 组成的有限序列，记作$(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$，这里的数据元素 a_i（$1\leqslant i\leqslant n$）只是一个抽象的符号，其具体含义在不同情况下可以不同，它既可以是原子类型，也可以是结构类型，但同一线性表中的数据元素必须属于同一数据对象。此外，线性表中相邻数据元素之间存在着序偶关系，即对于非空的线性表$(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$，表中 a_{i-1} 领先于 a_i，称 a_{i-1} 是 a_i 的直接前驱，而称 a_i 是 a_{i-1} 的直接后继。除了第一个元素 a_1 外，每个元素 a_i 有且仅有一个被称为其直接前驱的结点 a_{i-1}，除了最后一个元素 a_n 外，每个元素 a_i 有且仅有一个被称为其直接后继的结点 a_{i+1}。线性表中元素的个数 n 被定义为线性表的长度，n=0 时称为空表。

线性表的特点可概括如下。

- 同一性：线性表由同类数据元素组成，每一个 a_i 必须属于同一数据对象。
- 有穷性：线性表由有限个数据元素组成，表长度就是表中数据元素的个数。
- 有序性：线性表中相邻数据元素之间存在着序偶关系$<a_i,a_{i+1}>$。

由此可见，线性表是一种最简单的数据结构，因为数据元素之间是由一个前驱和一个后继的直观有序的关系确定；线性表又是一种最常见的数据结构，因为矩阵、数组、字符串、堆栈、队列等都符合线性条件。

1.1.1 线性表的顺序存储结构

线性表的顺序存储是指，用一组地址连续的存储单元依次存储线性表中的各个元素，使得线性表中在逻辑结构上相邻的数据元素存储在相邻的物理单元中，即通过数据元素物理存储的相邻关系反映数据元素之间逻辑结构上的相邻关系。采用顺序存储结构的线性表通常称为顺序表。

1.1.2 插入操作

线性表的插入运算是指在表的第 i（$1 \leqslant i \leqslant n+1$）个位置，插入一个新元素 a，使长度为 n 的线性表$(a_1,\cdots,a_{i-1},a_i,\cdots,a_n)$变成长度为 $n+1$ 的线性表$(a_1,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$。用顺序表作为线性表的存储结构时，由于结点的物理顺序必须和结点的逻辑顺序保持一致，因此必须将原表中位置 n，$n-1$，…，i 上的结点，依次后移到位置 $n+1$，n，…，$i+1$ 上，空出第 i 个位置。然后在该位置上插入新结点 a。当 $i=n+1$ 时，是指在线性表的末尾插入结点，所以无需移动结点，直接将 a 插入表的末尾即可。

例如，已知线性表(4,9,15,28,30,30,42,51,62)，需在第 4 个元素之前插入一个元素 21，则需要将第 4 个～第 9 个位置的元素依次后移一个位置，然后将 21 插入到第 4 个位置，如图 1-1 所示。请注意区分元素的序号和数组下标。

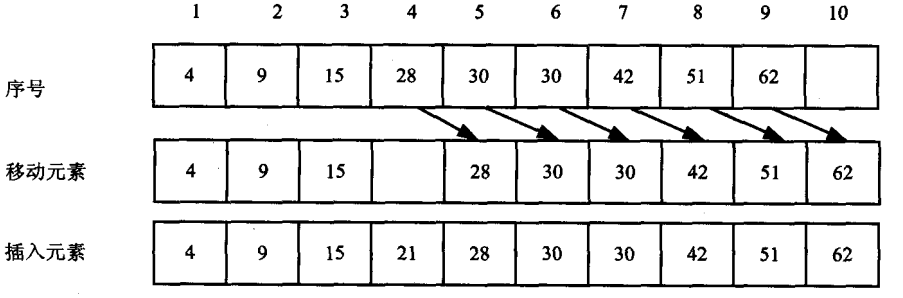


图 1-1　在线性表中移动和插入元素

1.1.3 删除操作

线性表的删除运算是指将表中的第 i（$1 \leqslant i \leqslant n$）个元素删去，使长度为 n 的线性表$(a_1,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$变成长度为 $n-1$ 的线性表$(a_1,\cdots,a_{i-1},a_{i+1},\cdots,a_n)$。

例如，删除线性表(4,9,15,21,28,30,30,42,51,62)中的第 5 个元素，则需将第 6 个～第 10 个元素依次向前移动一个位置，如图 1-2 所示。

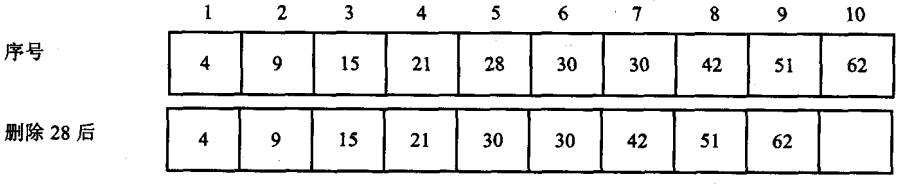


图 1-2　从线性表中删除元素

1.1.4 线性表的链式存储

为了克服顺序表的缺点，可以采用链式方式存储线性表，通常将采用链式存储结构的线性表称为链表。

1.1.5 单链表

在顺序表中，使用一组地址连续的存储单元依次存储线性表中的结点，因此结点的

逻辑次序和物理次序是一致的。而链表则不然，链表是用一组任意的存储单元来存储线性表中的结点，这组存储单元可以是连续的，也可以是不连续的，甚至是零散地分布在内存的任意位置上。因此，链表中结点的逻辑次序和物理次序不一定相同。为了正确地表示结点间的逻辑关系，必须在存储线性表中每个数据元素值的同时，存储指示其后继结点的地址（或位置）信息，这两部分信息组成的存储映像叫做结点（Node），如图 1-3 所示。

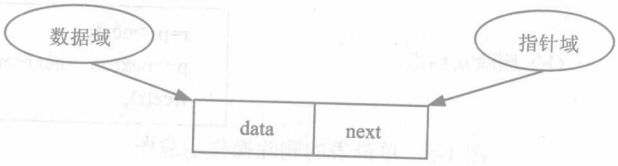
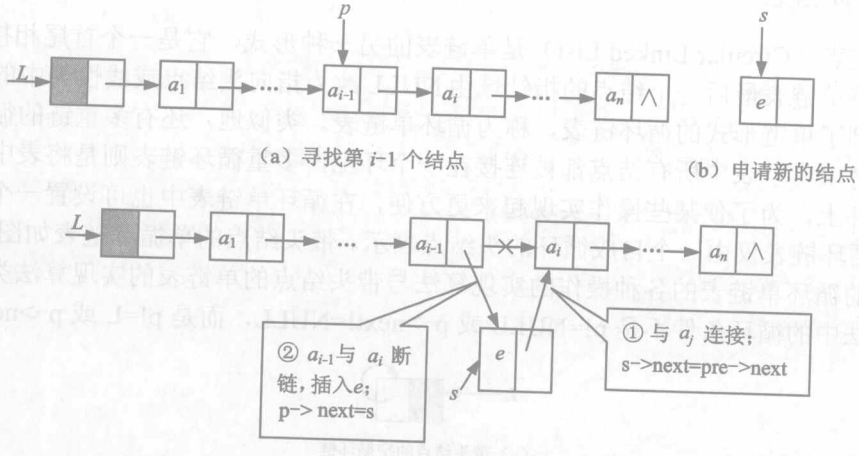


图 1-3　结点的组成信息

1.1.6　单链表的插入操作

算法描述：要在带头结点的单链表 L 中的第 i 个位置插入一个数据元素 e，首先需要在单链表中找到第 i-1 个结点并由指针 p 指示，然后申请一个新的结点并由指针 s 指示，其数据域的值为 e 并修改第 i-1 个结点的指针使其指向 s，然后使 s 结点的指针域指向原第 i 个结点，插入结点的过程如图 1-4 所示。



（a）寻找第 i-1 个结点　　（b）申请新的结点

（c）插入 a_i 结点

图 1-4　单链表的插入操作示意图

说明：当单链表中有 m 个结点时，插入位置有 m+1 个，即 $1 \leqslant i \leqslant m+1$。当 $i=m+1$ 时，认为是在单链表的尾部插入一个结点。

1.1.7　单链表的删除操作

算法描述：欲在带头结点的单链表 L 中删除第 i 个结点，则首先要通过计数方式找到第 i-1 个结点并使 p 指向第 i-1 个结点，然后删除第 i 个结点并释放结点空间。删除操作的过程如图 1-5 所示。

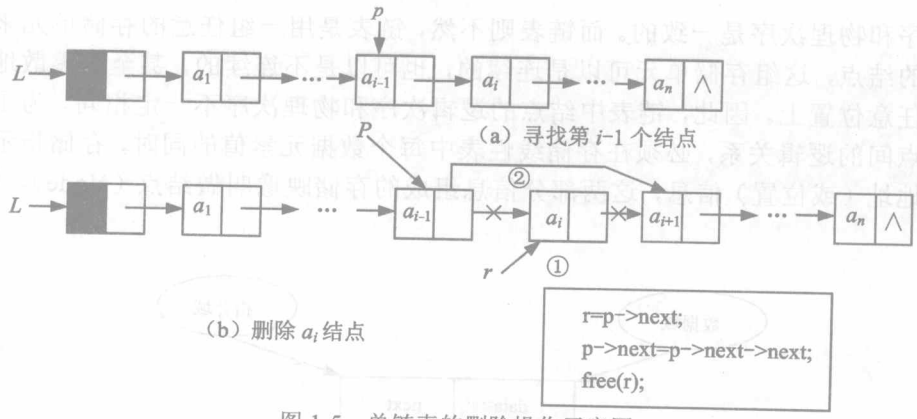


图 1-5　单链表的删除操作示意图

说明：删除算法中的循环条件（p->next! = NULL&&k<i-1）与插入算法中的循环条件（P!=NULL&&k<i-1）不同，因为插入时的插入位置有 m+1 个（m 为当前单链表中数据元素的个数）。i=m+1 是指在第 m+1 个位置前插入，即在单链表的末尾插入。而删除操作中删除的合法位置只有 m 个，若使用与插入操作相同的循环条件，则会出现指针指空的情况，使删除操作失败。

1.1.8　循环链表

循环链表（Circular Linked List）是单链表的另一种形式，它是一个首尾相接的链表，其特点是将单链表最后一个结点的指针域由 NULL 改为指向头结点或线性表中的第一个结点，就得到了单链形式的循环链表，称为循环单链表。类似地，还有多重链的循环链表。在循环单链表中，表中所有结点都被连接在一个环上，多重循环链表则是将表中的结点连接在多个环上，为了使某些操作实现起来更方便，在循环单链表中也可设置一个头结点，这样，空循环链表仅由一个自成循环的头结点表示。带头结点的单循环链表如图 1-6 所示，带头结点的循环单链表的各种操作的实现算法与带头结点的单链表的实现算法类似，差别仅在于算法中的循环条件不是 p!=NULL 或 p->next!=NULL，而是 p!=L 或 p->next!=L。

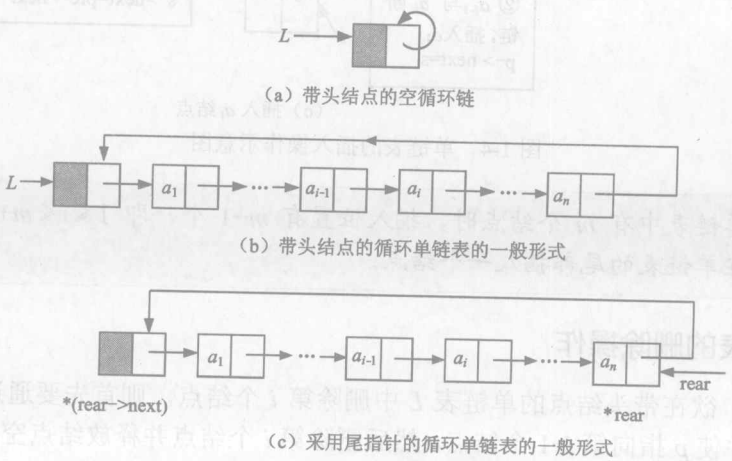


图 1-6　带头结点的单循环链表

1.1.9　双向链表

循环单链表的出现，虽然能够实现从任一结点出发沿着链能找到其前驱结点，但时间复杂度为 $O(n)$。如果希望从表中快速确定某一个结点的前驱，另一个解决方法就是在单链表的每个结点里再增加一个指向其前驱的指针域 prior。这样形成的链表中就有两条不同方向的链，称为双（向）链表（Double Linked List）。

1.1.10　双向链表的插入操作

算法描述：欲在双向链表的第 i 个结点之前插入一个新结点，则指针的变化情况如图 1-7 所示。

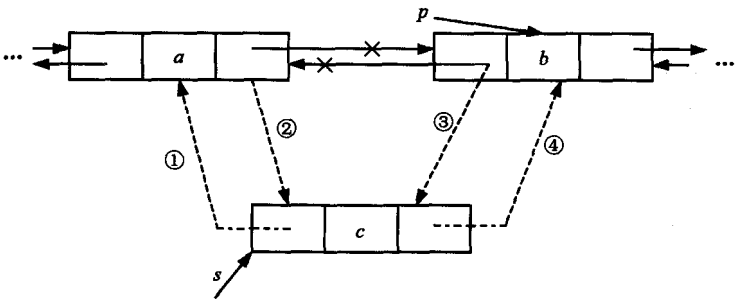


图 1-7　双向链表的插入操作示意图

1.1.11　双向链表的删除操作

算法描述：欲删除双向链表中的第 i 个结点，则指针的变化情况如图 1-8 所示。

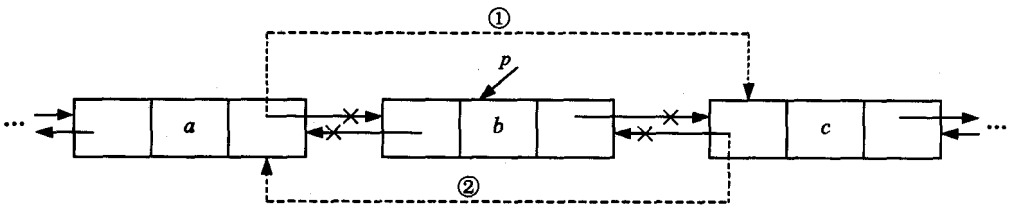


图 1-8　双向链表的删除操作示意图

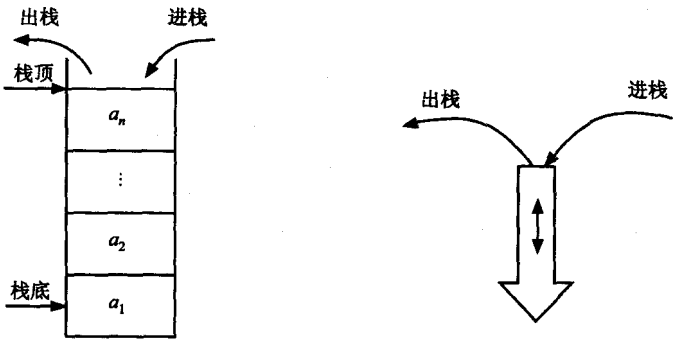
1.1.12　静态链表

以上介绍的各种链表都是由指针实现的，链表中结点空间的分配和回收（即释放）都是由系统提供的标准函数 malloc()和 free()动态的实现，故称为动态链表。但是有的高级语言，如 BASIC、FORTRAN 等，没有提供“指针”这种数据类型，此时若仍想采用链表作为存储结构，就必须使用“游标（Cursor）”来模拟指针，由程序员自己编写“分配结点”和“回收结点”的过程。

1.2　栈

栈作为一种限定性线性表，是将线性表的插入和删除运算限制为仅在表的一端进行，通常将表中允许进行插入、删除操作的一端称为栈顶（Top），因此栈顶的当前位置是动态

变化的，它由一个称为栈顶指针的位置指示器指示。同时表的另一端被称为栈底(Bottom)。当栈中没有元素时称为空栈。栈的插入操作被形象的称为进栈或入栈，删除操作称为出栈或退栈，栈的示意图如图 1-9 所示。



(a) 数据流进、出栈的示意图　　(b) 铁路调度栈的示意图

图 1-9　栈的示意图

1.2.1　顺序栈

顺序栈是用顺序存储结构实现的栈，即利用一组地址连续的存储单元依次存储自栈底到栈顶的数据元素，同时由于栈操作的特殊性，还必须附设一个位置指针 top（栈顶指针）动态地指示栈顶元素在顺序栈中的位置，如图 1-10 所示。通常以 top=-1 表示空栈。顺序栈的存储结构可用 C 语言中的一维数组来表示。

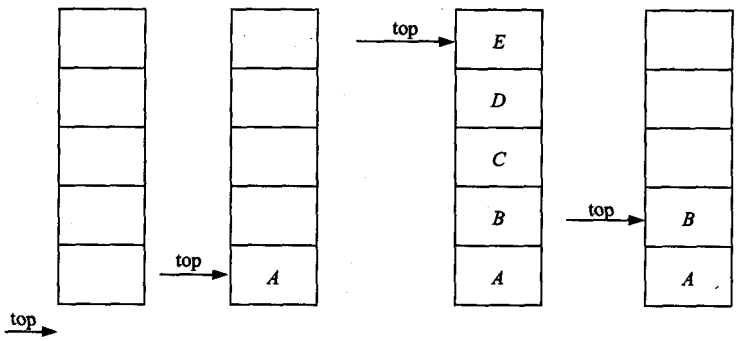


图 1-10　顺序栈的存储结构

顺序栈的基本操作如下。

1. 置栈空

```
void InitStack(SeqStack *S)
{   //将顺序栈置空
    S->top=-1;
}
```

2. 判栈空

```
int StackEmpty(SeqStack *S)
{
```

```
    return S->top==-1;
}
```

3. 判栈满

```
int StackFull(SeqStack *SS)
{
    return S->top==StackSize-1;
}
```

4. 进栈

```
void Push(S,x)
{
    if(StackFull(S))
       Error("Stack overflow");        //上溢，退出运行
    S->data[++S->top]=x;               //栈顶指针加1后将x入栈
}
```

5. 出栈

```
DataType Pop(S)
{
   if(StackEmpty(S)
      Error("Stack underflow");        //下溢，退出运行
   return S->data[S->top--];           //栈顶元素返回后将栈顶指针减1
}
```

6. 取栈顶元素

```
DataType StackTop(S)
{
  if(StackEmpty(S))
     Error("Stack is empty");
  return S->data[S->top];
}
```

栈的应用非常广泛，在一个程序中经常会出现需要同时使用多个栈的情况。若使用顺序栈，会因为对栈空间大小难以准确估计，从而产生栈溢出、栈空间空闲的情况。为了解决这个问题，可以让多个栈共享一个足够大的数组空间，通过利用栈的动态特性使其存储空间互相补充，这就是多栈的共享技术。

在栈的共享技术中最常用的是两个栈的共享技术：它主要利用了“栈的栈底位置不变，而栈顶位置动态变化”的特性。首先为两个栈申请一个共享的一维数组空间 $s[M]$，将两个栈的栈底分别存储在一维数组的两端，分别是 0，M-1。由于两个栈顶动态变化，这样可以形成互补，使得每个栈可用的最大空间与实际使用的需求有关，由此可见，两栈共享比两个栈分别申请 $M/2$ 的空间利用率高。两栈共享的数据结构定义如下：

```
#define M 100
Typedef struct
{
   StackElementType Stack[M];
```

```
    StackElementType top[2];              //top[0]和top[1]分别为两个栈顶指示器
}DqStack;
```

两个栈共享空间的示意图如图 1-11 所示，下面给出两个栈共用时的初始化、进栈和出栈操作的算法。

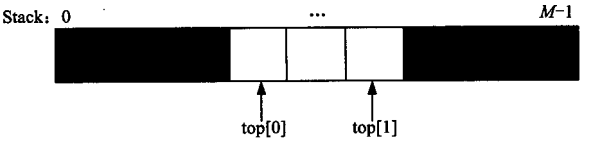


图 1-11　两个栈共享空间示意图

（1）初始化操作。

```
void InitStack(DqStack *S)
{
    S->top[0]=-1;
    S->top[1]=M;
}
```

（2）进栈操作算法。

```
int Push(DqStack *S, StackElementType x, int i)
{   //把数据元素 x 压入 i 号堆栈
   if(S->top[0]+1==S->top[1])        //栈已满
      return(FALSE);
   switch(i)
   {
      case 0:
          S->top[0]++;
          S->Stack[S->top[0]]=x;
          break;
      case 1:
          S->top[1]--;
          S->Stack[S->top[1]]=x;
          break;
          default:                   //参数错误
          return(FALSE)
   }
   return(TRUE);
}
```

（3）出栈操作算法。

```
int Pop(DqStack *S, StackElementType*x,int i)
{  //从 i 号堆栈中弹出栈顶元素并送到 x 中
   switch(i)
   {
       case 0:
           if(S->top[0]==-1)
```