第3章 特殊线性表

CHAPTER

3.1 内容概述

本章首先介绍栈的定义、特点、存储结构及其基本操作的实现，然后介绍队列的定义、特点、存储结构及其基本操作的实现，之后介绍串的定义、存储结构及其基本操作的实现，最后介绍两种串模式匹配算法。本章的知识结构如图 3.1 所示。

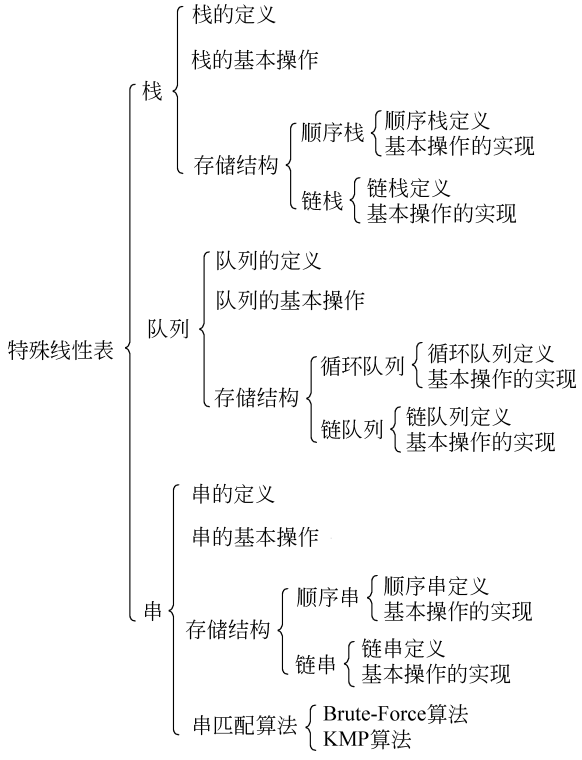

图 3.1 第 3 章知识结构

考核要求：掌握栈的特点及其基本操作的实现；掌握队列的特点及其

基本操作的实现；掌握串的特点及其基本操作的实现；掌握 Brute-Force 和 KMP 两种串模式匹配算法。

重点难点：本章的重点是栈、队列和串的特点，栈、队列和串的存储结构及基本操作的实现，串的两种模式匹配算法。本章的难点是如何使用栈和队列处理实际问题，循环队列中对边界条件的处理以及串的模式匹配算法。

核心考点：栈和队列的特点，栈和队列的存储结构及其基本操作的实现，利用栈和队列设计解决具体问题的算法，串的概念、各种运算及串的模式匹配算法。

3.2 典型题解析

3.2.1 栈的特点及基本操作

栈的特点是后进先出。栈的基本操作主要是进栈和出栈，进栈时要判断栈是否为满，出栈时要判断栈是否为空。

【例 3.1】 一个栈的输入序列为 12345，则下列序列中不可能是栈的输出序列的是（　　）。

A. 23415　　B. 54132　　C. 23145　　D. 15432

解析：输入序列为 12345，输出序列不可能是 54132，理由是输出序列的第一个元素是 5，输出的序列一定是 54321。得到 23415 的过程如下：1、2 入栈，2 出栈，得到部分输出序列 2；3 入栈并出栈，得到部分输出序列 23；4 入栈并出栈，得到部分输出序列 234；1 出栈，得到部分输出序列 2341；5 入栈并出栈，得最终结果为 23415。得到 23145 的过程如下：1、2 入栈，2 出栈，得到部分输出序列 2；3 入栈并出栈，得到部分输出序列 23；1 出栈，得到部分输出序列 231；4 入栈并出栈，得到部分输出序列 2314；5 入栈并出栈，得最终结果为 23415。得到 15432 的过程如下：1 入栈并出栈，得到部分输出序列 1；2、3、4、5 依次入栈，然后出栈，得最终结果为 15432。

答案：B

【例 3.2】 若一个栈的输入序列为 1，2，3，…，n，且输出序列的第一个元素是 i，则第 j 个输出元素是（　　）。

A. i－j＋1　　B. i－j　　C. j－i＋1　　D. 不确定

解析：对输入序列 1，2，3，…，n，若第一个输出的元素是 n，即 i＝n，则第 j 个输出的元素是 n－j＋1，否则不能确定其他元素的输出顺序。例如：输入序列为 123，若第一个输出元素是 1，则可能的输出序列有 123 和 132。

答案：D

【例 3.3】 若栈采用顺序存储方式存储，现两个栈共享空间 V[0..m－1]，top[i]代表第 i 个栈(i＝1，2)的栈顶，栈 1 的底在 v[0]，栈 2 的底在 V[m－1]，则栈满的条件是（　　）。

A. |top[2]－top[1]|＝＝0　　B. top[1]＋1＝＝top[2]

C. top[1]＋top[2]＝＝m　　D. top[1]＝＝top[2]

解析：栈 1 和栈 2 共享内存中的一片连续空间，两个栈顶 top[1]和 top[2]向共享空间的中心延伸，仅当两个栈顶指针值之差的绝对值等于 1 时为栈满。

答案：B

【例 3.4】 两个不同的合法输入序列能否得到相同的输出元素序列？如能得到，请举例说明。

解析：可以得到相同的输出元素序列。例如，输入元素为 A、B、C，则两个输入序列 ABC 和 BAC 均可得到输出元素序列 ABC。对于合法序列 ABC，使用 S×S×S×操作序列；对于合法序列 BAC，使用 SS××S×操作序列（其中，S 表示入栈，×表示出栈）。

【例 3.5】 假设以 I 和 O 分别表示入栈和出栈操作。栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，称可以操作的序列为合法序列，否则称为非法序列。

（1）下列所示的序列中合法的有（　　）。

A. IOIIOIOO　　B. IOOIOIIO　　C. IIIOIOIO　　D. IIIOOIOO

（2）通过对（1）的分析，写出一个算法，判定所给的操作序列是否合法。若合法，则返回 1，否则返回 0（假设被判定的操作序列已存入一维数组中）。

解析：在入栈和出栈序列（即由'I'和'O'组成的字符串）的任一位置，入栈次数（'I'的个数）都必须大于或等于出栈次数（即'O'的个数），否则视为非法序列。整个序列的入栈次数必须等于出栈次数，否则视为非法序列。

（1）A 和 D 是合法序列，B 和 C 是非法序列。

（2）设被判定的操作序列已存入一维数组 A。

```
int Judge(char A[])
{ int i=0;                              /*i 为数组 A 的下标*/
  int j=0,k=0;                          /*j 和 k 分别为字母 I 和 O 的个数*/
  while(A[i]!='\0')
  { switch(A[i])
    { case 'I': j++; break;             /*入栈次数增 1*/
      case 'O': k++; if(k>j) return 0;/*出栈次数大于入栈次数*/
    }
    i++;
  }
  if(j!=k) return 0;                    /*整个入栈次数不等于出栈次数*/
  else return 1;
}
```

【例 3.6】 若借助栈由输入序列 1，2，…，n 得到输出序列为 P_1，P_2，…，P_n（它是输入序列的一个排列），试问是否存在 i<j<k，使得 $P_j<P_k<P_i$ 成立。

解析：如果 i<j，则对于 $P_i<P_j$ 的情况，说明 P_i 在 P_j 入栈前已先出栈。而对于 $P_i>P_j$ 的情况，则说明要将 P_j 压到 P_i 之上，也就是在 P_j 出栈之后 P_i 才能出栈。这就说明，对于 i<j<k，不可能出现 $P_j<P_k<P_i$ 的输出序列。例如，对于输入序列 123，不可能出现输出序列 312。

【例 3.7】 设整数序列 $a_1,a_2,\cdots,a_n$，给出求解最大值的递归程序。

解析：利用递归法求解数组 a 中 n 个元素的最大值的计算公式为

$$\max(a,n)=\begin{cases}a[0] & n=1\\ a[n-1]>\max(a,n-1)?\ a[n-1]:\max(a,n-1) & n>1\end{cases}$$

n=1 为递归结束条件。

```
int MaxValue(int a[],int n)
{ int max;
  if(n==1) max=a[0];
  else if(a[n-1]>MaxValue(a,n-1)) max=a[n-1];
       else max=MaxValue(a,n-1);
  return max;
}
```

3.2.2 队列的特点及基本操作

队列的特点是先进先出，考查方式通常是入队时队头指针的修改以及出队时队尾指针的修改。队列的基本操作主要考查入队和出队，出队时要判断队列是否为空，入队时要判断队列是否已满。另外，考查综合利用栈和队列实现某些具体操作的能力。

【例 3.8】 若用一个大小为 m 的数组实现循环队列，队头指针 front 指向队头元素，队尾指针 rear 指向队尾元素的下一个位置，则当前队列中的元素个数是（　　）。

A. (rear－front＋m)％m　　B. rear－front＋1

C. rear－front－1　　D. rear－front

解析：根据循环队列的特点，有以下几种情况：

$$\text{队列中元素个数}=\begin{cases}0 & \text{front}=\text{rear}\\ \text{rear}-\text{front} & \text{rear}>\text{front}\\ \text{rear}-\text{front}+m & \text{rear}<\text{front}\end{cases}$$

归纳起来，循环队列中元素个数的计算公式为

$$(\text{rear}-\text{front}+m)\%m$$

答案：A

【例 3.9】 用一个大小为 6 的数组实现循环队列，队头指针 front 指向队头元素，队尾指针 rear 指向队尾元素的下一个位置，当前 rear 和 front 的值分别为 0 和 3。从队列中删除两个元素，再加入两个元素后，rear 和 front 的值分别为（　　）。

A. 1 和 5　　B. 2 和 5　　C. 4 和 2　　D. 5 和 1

解析：元素出队时，front 的计算公式为 front＝(front＋1)％6，出队两个元素，front 的值为 5。元素入队时，rear 的计算公式为 rear＝(rear＋1)％6，入队两个元素，rear 的值为 2。

答案：B

【例 3.10】 若用不带头结点的单链表存储队列，其队头指针指向队头结点，其队尾指针指向队尾结点，则在进行删除操作时（　　）。

A. 仅修改队头指针　　B. 仅修改队尾指针

C. 队头、队尾指针都要修改　　D. 队头、队尾指针都可能要修改

解析：当队列中有两个或两个以上的结点时，删除操作只修改队头指针；当队列中只有一个结点时，删除操作既要修改队头指针，又要修改队尾指针。

答案：D

【例 3.11】 设栈 S 和队列 Q 的初始状态都为空，元素 e1、e2、e3、e4、e5 和 e6 依次通过栈 S，一个元素出栈后即进入队列 Q，若 6 个元素的出队序列是 e2、e4、e3、e6、e5、e1，则栈 S 的容量至少应该是(　　)。

A. 6　　B. 4　　C. 3　　D. 2

解析：出队序列就是出栈序列，得到出栈序列 e2、e4、e3、e6、e5、e1 的过程是：e1、e2 依次进栈，栈中有 2 个元素，e2 出栈，栈中有 1 个元素；e3、e4 依次进栈，栈中有 3 个元素，e4 出栈，e3 出栈，栈中有 1 个元素；e5、e6 依次进栈，栈中有 3 个元素，e6 出栈，e5 出栈，栈中有 1 个元素；e1 出栈，栈空。由此可知，栈中最多有 3 个元素，所以栈 S 的容量至少应该是 3。

答案：C

【例 3.12】 已知栈和队列的基本操作如下。

InitStack(&s)：初始化空栈 s。

Push(&s,x)：元素 x 入 s 栈。

Pop(&s)：s 栈顶元素出栈，并返回栈顶元素。

EmptyStack(&s)：判 s 栈是否为空，空则返回 1，否则返回 0。

EnQueue(q,x)：元素 x 入队列。

OutQueue(q)：队头元素出队列，并返回队头元素。

EmptyQueue(q)：判队列为空。

简述算法 fun 的功能。

```
void Fun(CirQueue *q)
{ SeqStack s;
  InitStack(&s);
  while(!EmptyQueue(q))
    Push(&s,OutQueue(q));
  while(!EmptyStack(&s))
    EnQueue(q,Pop(&s));
}
```

解析：函数 fun 中第一个循环完成的功能是将队列中的全部元素出队并入栈，第二个循环完成的功能是将栈中的全部元素出栈并入队列。由于栈的特点是后进先出，因此队列中元素的顺序与原顺序相反。由此可知，函数 fun 的功能是将队列中的元素逆置存放。

【例 3.13】 已知栈和队列的基本操作如下。

InitStack(&s)：初始化空栈 s。

Push(&s,x)：元素x入s栈。

Pop(&s)：s栈顶元素出栈，并返回栈顶元素。

EmptyStack(&s)：判s栈是否为空，空则返回1，否则返回0。

InitQueue(&q)：初始化空队列q。

EnQueue(q,x)：元素x入队列。

OutQueue(q)：队头元素出队列，并返回队头元素。

EmptyQueue(q)：判队列为空。

设栈s=(1,2,3,4,5,6,7)，其中7为栈顶元素，写出调用下列函数fun后的s。

```
void Fun(SeqStack *s)
{ CirQueue q; SeqStack t; int i=0;
  InitQueue(&q);
  InitStack(&t);
  while(!EmptyStack(s))
    if((i=! i)!=0) Push(&t,Pop(s));
    else EnQueue(&q, Pop(s));
  while(!EmptyStack(&t))
    Push(s,Pop(&t));
  while(!EmptyQueue(&q))
    Push(s,OutQueue(&q));
}
```

解析：函数fun中第一个循环完成的功能是将栈s中的7、5、3、1出栈后入栈t，6、4、2出栈后入队列q；第二个循环的完成功能是将栈t中的1、3、5、7出栈后入栈s；第三个循环的完成功能是将队列q中的6、4、2出队后入栈s。由此可知，调用函数fun后s=(1,3,5,7,6,4,2)。

【例3.14】 如果用一维数组q[0..m－1]表示循环队列，该队列只有一个队列头指针front(指向队头元素)，不设队列尾指针rear，而是设置计数器count，用来记录队列中结点的个数。编写实现队列的三个基本运算的算法：判空、入队和出队。

解析：队空时，count的值为0。队满时，count的值为m。队尾指针为(front＋count)%m。入队时，要判断队列是否为满。出队时，要判断队列是否为空。

```
typedef int ElemType;
typedef struct
{ ElemType q[m];
  int front,count;          /*front是队头指针,count是队列中的元素个数*/
}CirQueue;                  /*循环队列类型*/
```

① 判空。

```
int EmptyQueue(CirQueue *Q)
{ if(Q->count==0) return 1;
  else return 0;
}
```

② 入队。

```
int EnQueue(CirQueue *Q,ElemType x)
{ if(Q->count==m) { printf("队满\n");return 0;}          /*队满*/
  Q->q[(Q->front+Q->count)%m]=x;                        /*入队*/
  Q->count++;                                           /*元素个数加 1*/
  return 1;
}
```

③ 出队。

```
int OutQueue(CirQueue *Q,ElemType *e)
{ if(Q->count==0){printf("队空\n");return 0;}           /*队空*/
  *e=Q->q[Q->front];                                    /*保存对头元素*/
  Q->front=(Q->front+1)%m;                              /*修改对头指针*/
  Q->count--;                                           /*元素个数加 1*/
  return 1;
}
```

【例 3.15】 试用一个指针 p 和某种链表结构实现一个队列。设结点结构为(data，next)，给出入队 EnQueue 和出队 OutQueue 的过程，要求它们的时间复杂度都是 O(1)。

解析：用只设尾指针的单循环链表实现队列。在出队算法中，首先要判断队列是否为空。另外，出队后要判断是否因出队而成为空队列，否则可能导致因出队将尾指针删除而成为“悬挂变量”。

```
typedef int ElemType;
typedef struct node
{ ElemType data;
  struct node *next;
}slink;
```

① 入队。

```
slink *EnQueue(slink *p,ElemType x)     /*p 指向队尾结点*/
{ slink *s;
  s=(slink *)malloc(sizeof(slink));     /*申请新结点*/
  s->data=x;
  s->next=p->next; p->next=s;           /*将 s 结点入队*/
  p=s;
  return p;                             /*指针 p 移至新的队尾*/
}
```

② 出队。

```
slink *OutQueue(slink *p, ElemType *e) /*p 指向队尾结点*/
{ slink *s;
  if(p->next==p) return NULL;           /*带头结点的空循环队列*/
```

```
    s=p->next->next;                    /*找到队头元素*/
    p->next->next=s->next;              /*删除队头元素*/
    *e=s->data;                         /*返回出队元素*/
    if(p==s) p=p->next;                 /*队列中只有一个结点,出队后成为空队列*/
    free(s);
    return p;
}
```

3.2.3 串的有关概念及基本操作

串的特点是数据元素是字符，考查内容主要包括三个方面：一是串的有关概念，如串、子串、空串和空格串等；二是根据要求对字符串进行处理，如调整字符位置，将数字串转换成对应的数值等；三是串的模式匹配算法，如计算串的 next 值和 nextval 值，在串中查找或处理满足给定条件的子串等。

【例 3.16】 下列关于串的叙述中不正确的是（　　）。

A. 串是字符的有限序列

B. 空串是由空格构成的串

C. 模式匹配是串的一种重要运算

D. 串既可以采用顺序存储，也可以采用链式存储

解析：串又称字符串，是由零个或多个字符组成的有限序列。串中的字符个数称为串长，含有 0 个字符的串称为空串。由空格构成的串是空格串，其长度为空格的个数。串有顺序存储和链式存储两种存储结构，分别称为顺序串和链串。模式匹配是串的一种重要运算。

答案：B

【例 3.17】 若串 S="structure"，则其子串的数目是（　　）。

A. 9　　B. 46　　C. 45　　D. 10

解析：子串是由串中任意个连续的字符组成的子序列，并规定空串是任意串的子串，任意串是其自身的子串。若字符串长度为 n(n>0)，则长度为 n 的子串有 1 个，长度为 n−1 的子串有 2 个，长度为 n−2 的子串有 3 个，……，长度为 1 的子串有 n 个。由此可知，长度为 n 的字符串的子串数为 n(n+1)/2+1。

答案：B

【例 3.18】 设字符串 S="aabaabaabaac"，P="aabaac"。

(1) 给出 S 和 P 的 next 值和 nextval 值。

(2) 若 S 作为主串，P 作为模式串，试给出利用 Brute-Force 算法和 KMP 算法的匹配过程。

解析：next 的计算方法如下。

```
#define INITSIZE 100                    /*为串分配的存储空间的初始量*/
typedef struct
{ char *ch;                             /*串存放的起始地址*/
  int length;                           /*串长*/
  int strsize;                          /*当前为串分配的存储空间*/
```

```
}SeqStr;
void GetNext(SeqStr * t,int next[])   /* 由模式串 t 求出 next 值 */
{ int j,k;
  j=0;k=-1;next[0]=-1;
  while(j<t->length)
    if(k==-1||t->ch[j]==t->ch[k])
    { j++;k++;next[j]=k;}
    else k=next[k];
}
```

由 next 计算 nextval 的方法如下。

如果 t－>ch[j]==t－>ch[k],则 nextval[j]=nextval[k],否则 nextval[j]=k。

S 的 next 和 nextval 值如下。

j	0	1	2	3	4	5	6	7	8	9	10	11
S 串	a	a	b	a	a	b	a	a	b	a	a	c
next[j]	－1	0	1	0	1	2	3	4	5	6	7	8
nextval[j]	－1	－1	1	－1	－1	1	－1	－1	1	－1	－1	8

P 的 next 和 nextval 值如下。

j	0	1	2	3	4	5
P 串	a	a	b	a	a	c
next[j]	－1	0	1	0	1	2
nextval[j]	－1	－1	1	－1	－1	2

利用 Brute-Force 算法的匹配过程如下。

第一趟匹配：aabaabaabaac
　　aabaac(i=6,j=6)

第二趟匹配：aabaabaabaac
　　aa(i=3,j=2)

第三趟匹配：aabaabaabaac
　　a(i=3,j=1)

第四趟匹配：aabaabaabaac
　　aabaac(i=9,j=6)

第五趟匹配：aabaabaabaac
　　aa(i=6,j=2)

第六趟匹配：aabaabaabaac
　　a(i=6,j=1)

第七趟匹配：aabaabaabaac
（成功）　　aabaac(i=13,j=7)

利用 KMP 算法的匹配过程如下。

第一趟匹配：aabaabaabaac
　　aabaac(i=6,j=6)

第二趟匹配：aabaabaabaac
　　(aa)baac

第三趟匹配：aabaabaabaac
（成功）　　(aa)baac

【例3.19】 已知串处理函数如下。

StrLen(char *s)：返回串s的长度。

Index(char *st,char *t)：返回串t在串st中首次出现的下标值，若不存在，则返回－1。

简述下列函数fun的功能。

```
int Fun(char *s,char *t,int pos[])
{ int i,j,k,ls,lt;
  ls=StrLen(s);
  lt=StrLen(t);
  if(ls==0||lt==0) return -1;
  k=0; i=0;
  do
  { j=Index(s+i,t);
    if(j>=0)
    { pos[k++]=i+j;
      i+=j+lt;
    }
  }while(i+lt<=ls&&j>=0);
  return  k;
}
```

解析：函数index(s＋i,t)的功能是返回串t在串s＋i中首次出现的下标值，函数fun中循环语句的功能是查找串t在串s中每次出现的下标值，并按递增顺序存放到数组pos中，用k记录串t在串s中出现的次数。所以函数fun的功能是统计串t在串s中出现的次数，并将每次在s中出现的下标值按增序存放到数组pos中。

【例3.20】 输入一个字符串，其中有数字和非数字字符。编写算法，将其中的所有数字子串转换为其对应的数值并依次存放到一维数组a中，同时统计数字子串的个数并输出这些数。例如，对于字符串"ak123x456&.179?302gef4563"，将123放入a[0]，456放入a[1]，179放入a[2]，302放入a[3]，4563放入a[4]，数字子串的个数为5。

解析：从左到右扫描字符串，初次遇到数字字符时作为一个整数的开始，然后进行转换，即将连续出现的数字字符转换为一个整数，直到非数字字符为止。一个整数转换完成后存入数组，再准备下一个整数，如此下去，直至整个字符串扫描结束。

```
int Count(char s[],int a[])
{ int i=0,num,k=0;                    /*num暂存转换结果,k存放数字子串的个数*/
  while(s[i]!='\0')
  { if(s[i]>='0'&&s[i]<='9')
    { num=0;                          /*初始化*/
      while(s[i]>='0'&&s[i]<='9')     /*转换*/
      { num=num*10+s[i]-'0';
        i++;
      }
```

```
      a[k++]=num;                    /* 转换结果存入数组 */
    }
    else i++;
  }
  return k;
}
```

【例 3.21】 以顺序存储结构表示串，设计算法，求串 S 中出现的第一个最长重复子串及其位置，并分析算法的时间复杂度。

解析：设以字符数组 s 表示串，重复子串的含义是由一个或多个连续相同的字符组成的子串，其长度用 max 表示，初始长度为 0，将每个局部重复子串的长度与 max 相比，若比 max 大，则更新 max，并用 index 记住其开始位置。

```
int LongStr(char s[],int *index)   /* 通过形参带回最长重复子串的开始位置 */
{ int max=0,n;
  int length=1,i=0,start=0;
  *index=0;
  n=strlen(s);
  while(i<n-1)
    if(s[i]==s[i+1]) {i++;length++;}
    else
    { if(max<length) { max=length; *index=start;}
      i++;start=i;length=1;
    }
  return max;                      /* 返回最长重复子串的长度 */
}
```

每个字符与其后继比较一次，算法的时间复杂度为 O(n)。

【例 3.22】 S="$S_1S_2\cdots S_n$"是一个长为 n 的字符串，存放在一维数组中。编写算法，将 S 中所有下标为偶数的字符按其原来下标从大到小的顺序放在 S 的后半部分，所有下标为奇数的字符按其原来下标从小到大的顺序放在 S 的前半部分。

例如：若 S="ABCDEFGHIJKL"，则改造后的 S 为"ACEGIKLJHFDB"。

解析：从头向尾扫描字符串，下标为奇数的字符直接放在数组前面，下标为偶数的字符入栈，扫描结束后，再将栈中的字符出栈并送入数组。

```
void MovStr(char *s)
{ char stk[81];                    /* stk 是字符栈 */
  int i=0,j=0,k=0;                 /* i 是遍历字符串的下标变量,k 是字符栈指针 */
  while(s[i])                      /* 改造字符串 */
  { if((i+1)%2==1) s[j++]=s[i];
    else stk[k++]=s[i];
    i++;
  }
  k--;
```

```
    while(k>=0) s[j++]=stk[k--];     /*将下标为偶数的字符逆序填入原字符数组*/
}
```

3.3 自测试题

1. 单项选择题

(1) 栈和队列的共同点是(　　)。

A. 都是先进先出　　B. 都是先进后出

C. 只允许在端点处插入和删除元素　　D. 没有共同点

(2) 一个栈的输入序列为1,2,3,…,n,若输出序列的第一个元素是n,则输出的第i(1≤i≤n)个元素是(　　)。

A. 不确定　　B. n−i+1　　C. i　　D. n−i

(3) 若6个元素6、5、4、3、2、1顺序进栈,则(　　)不是合法的出栈序列。

A. 543612　　B. 453126　　C. 346521　　D. 234156

(4) 若一个栈以向量V[1..n]存储,初始栈顶指针top为n+1,则x进栈的正确操作是(　　)。

A. top=top+1;V[top]=x　　B. V[top]=x;top=top+1

C. top=top−1;V[top]=x　　D. V[top]=x;top=top−1

(5) 假设以数组A[m]存放循环队列的元素,其头尾指针分别为front和rear,则当前队列中的元素个数为(　　)。

A. (rear−front+m)%m　　B. rear−front+1

C. (front−rear+m)%m　　D. (rear−front)%m

(6) 设计一个判别表达式中左右括号是否配对出现的算法时,采用(　　)数据结构最佳。

A. 线性表的顺序存储结构　　B. 队列

C. 线性表的链式存储结构　　D. 栈

(7) 串的长度是指(　　)。

A. 串中所含不同字母的个数　　B. 串中所含字符的个数

C. 串中所含不同字符的个数　　D. 串中所含非空格字符的个数

(8) 串"ababaaababaa"的next数组值为(　　)。

A. −101234567888　　B. −101010000101

C. −100123112345　　D. −101201211234

(9) 字符串"ababaabab"的nextval数组值为(　　)。

A. −1,0,−1,0,−1,4,0,−1,0　　B. −1,0,−1,0,−1,1,0,−1,0

C. −1,0,−1,0,−1,−1,−1,0,0　　D. −1,0,−1,0,−1,0,−1,0,0

(10) 设S为一个长度为n的字符串,其中的字符各不相同,则S中互异的非平凡子串(非空且不同于S本身)的个数为(　　)。

A. 2n−1　　B. n^2　　C. n(n+1)/2　　D. n(n+1)/2−1

2. 正误判断题

(1) 栈是限定仅在表尾进行插入和删除操作的线性表。　(　)
(2) 引入循环队列的目的是避免假溢出时大量移动数据元素。　(　)
(3) 区分循环队列的满与空,有牺牲一个存储单元和设标记两种方法。　(　)
(4) 若一个栈的输入序列为 1、2、3,则 312 是不可能的栈输出序列。　(　)
(5) 表达式求值是队列应用的一个典型例子。　(　)
(6) 任何一个递归过程都可以转换成非递归过程。　(　)
(7) 循环队列也存在空间溢出问题。　(　)
(8) 栈和队列都是线性表,只是在插入和删除时受到了一些限制。　(　)
(9) KMP 算法的特点是在模式匹配时指示主串的指针不会变小。　(　)
(10) 串是一种数据对象和操作都特殊的线性表。　(　)

3. 填空题

(1) 用 S 表示入栈操作,X 表示出栈操作。若元素入栈的顺序为 1234,则为了得到 1342 的出栈顺序,相应的 S 和 X 的操作串为 ① 。

(2) 用 data[0..n−1]表示一个顺序栈,栈顶指针是 top,则将值为 x 的元素入栈的操作是 ② 。

(3) 用下标从 0 开始的 N 元数组实现循环队列时,为实现下标变量 M 加 1 后在数组有效下标范围内循环,可采用的表达式是 M= ③ 。

(4) 设循环队列存放在向量 sq.data[0..M]中,若用牺牲一个单元的办法区分队满和队空(设队首指针为 sq.front,队尾指针为 sq.rear),则队满的条件为 ④ 。

(5) 设目标串长度为 n,模式串长度为 m,则串匹配的 KMP 算法的时间复杂度为 ⑤ 。

(6) 模式串 P="abaabcac"的 next 数组值为 ⑥ 。

(7) 字符串"ababaaab"的 nextval 数组值为 ⑦ 。

(8) 设 T 和 P 是两个给定的串,在 T 中寻找等于 P 的子串的过程称为 ⑧ 。

(9) 如果希望循环队列中的向量单元都能得到利用,则可设置一个标志域 tag,每当尾指针和头指针的值相同时,以 tag 的值为 0 或 1 区分队列的状态是“空”还是“满”。请为下列函数填空,使其分别实现与此结构相对应的入队列和出队列的算法。

```
#define MAXSIZE 100  /* 队列空间的初始分配量 */
typedef int ElemType;/* 数据元素类型 */
typedef struct
{ ElemType *data;    /* 基地址 */
  int front;         /* 队头指针 */
  int rear;          /* 队尾指针 */
}CirQueue;
int tag;             /* 当尾指针和头指针值相同时,若 tag=1 则队满;若 tag=0 则队空 */
int EnQueue(CirQueue *q,ElemType x)
{ if( ⑨ ) return 0;
```

```
  q->data[q->rear]=x;
  q->rear=(q->rear+1)%MAXSIZE;
   ⑩ ;
  return 1;
}
int OutQueue(CirQueue *q, ElemType *x)
{ if( ⑪ ) return 0;
  *x=q->data[q->front];
  q->front= ⑫ ;
  ⑬ ;
  return 1;
}
```

（10）下列函数的功能是判断字符串s是否对称，若对称则返回1，否则返回0。例如，Judge("abba")返回1，Judge("abab")返回0。

```
int Judge( ⑭ )
{ int i=0,j=0;
  while(s[j]) ⑮ ;
  for(j--;i<j&&s[i]==s[j];i++,j--);
  return( ⑯ );
}
```

4. 算法设计题

（1）设表达式以字符形式已存入数组E，'#'为表达式的结束符，试写出判断表达式中括号'('和')'是否配对的算法（算法中可调用栈操作的基本算法）。

（2）如果允许在循环队列的两端都可以进行插入和删除操作，请写出循环队列的类型定义，并写出从队尾删除和从队头插入的算法。

（3）设计一个算法，将一个整型数字串转换为其对应的数值。

3.4 实验题

（1）编写算法，利用栈将一维整型数组a中大于0的元素存放在数组的前面，小于或等于0的元素存放在数组的后面。

（2）编写算法，利用栈计算算术表达式4+2×3-9/5的值。

（3）已知q是一个非空队列，s是一个空栈，编写算法，将队列q的内容逆置。该队列存储在一个一维数组中。

（4）编写算法，将一个链队列拆分成两个链队列，其中一个链队列的值都为偶数，另一个链队列的值都为奇数。假设链队列中数据域的值为整型数。

（5）编写算法，查找顺序串t在顺序串s中出现的次数。

（6）编写算法，删除链串s中的所有子串t。

（7）线性表中的元素存放在数组A[0..n-1]中，元素是整型数。试写出求A中最大

元素和最小元素的递归算法。

(8) 请利用两个栈 S1 和 S2 模拟一个队列,已知栈的三个运算定义如下。

Push(S,x):元素 x 入 S 栈。

Pop(S,x):S 栈顶元素出栈,赋给变量 x。

EmptyStack(S):判 S 栈是否为空。

编写利用栈的运算实现队列的入队、出队和判队空三个运算。

(9) 设 s、t 为两个字符串,分别放在两个一维数组中,其长度分别为 m 和 n。编写算法,判断 t 是否为 s 的子串。如果是,则输出子串所在的位置(第一个字符在 s 中的位序),否则输出 0。

(10) 两个整数序列 $A=(a_1,a_2,a_3,\cdots,a_m)$和 $B=(b_1,b_2,b_3,\cdots,b_n)$已经存入两个单链表中,设计一个算法,判断序列 B 是否是序列 A 的子序列。

3.5　思考题

(1) 链栈中是否可以不设头结点?

(2) 何谓队列的假溢出现象?解决的方法有哪些?

(3) 简述串属于线性表的理由。

(4) 两个字符串 S1 和 S2 的长度分别为 m 和 n,求这两个字符串最大共同子串的算法的时间复杂度为 T(m,n)。请估算最优的 T(m,n),并简要说明理由。

(5) 设主串 S="xxyxxxyxxxxyxyx",模式串 T="xxyxy"。请问:如何用最少的比较次数找到 T 在 S 中出现的位置?相应的比较次数是多少?

3.6　主教材习题解答

1. 单项选择题

(1) 在一个链队列中,假设 front 和 rear 分别为队首指针和队尾指针,则进行插入 s 所指向结点的操作是(　　)。

① front->next=s;front=s;　　② rear->next=s;rear=s;

③ front=front->next;　　④ front=rear->next;

解答:队列是一种特殊的线性表,它只允许在一端进行插入操作,在另一端进行删除操作。允许插入的一端称为队尾,允许删除的一端称为队头,新插入的结点只能添加到队尾,要删除的结点只能是排在队头的结点。在队尾指针为 rear 的链队列中插入 s 所指向结点的操作是

```
rear->next=s;rear=s;
```

答案:②

(2) 在具有 n 个单元的顺序存储的循环队列中,假设 front 和 rear 分别为队首指针和队尾指针,则判断队列为空的条件是(　　)。

① front==rear+1　　② front+1==rear

③ front==rear　　④ front==0

解答：当循环队列呈空状态时，有front==rear。为了区分循环队列的空满状态，解决的方法之一是少用一个元素空间，规定当(rear+1)%n==front时为循环队列满，即当队尾指针指向队头元素的上一个位置时就认为队列已满。

答案：③

(3) 串是由(　　)。

① 一些符号构成的序列　　② 一些字母构成的序列

③ 一个以上的字符构成的序列　　④ 任意有限个字符构成的序列

解答：串又称字符串，是一种特殊的线性表，其特殊性体现在表中的每个数据元素都是一个字符，即串是由0个或多个字符组成的有限序列。

答案：④

(4) 设输入序列为1234，借助一个栈可以得到的输出序列是(　　)。

① 1342　　② 3142　　③ 4312　　④ 4123

解答：输入序列为1234，输出序列不能是3142，理由是输出序列的第一个元素是3，一定是1、2、3依次入栈后3出栈，输出的第二个元素一定不是1。输出序列不可能是4312，也不可能是4123，理由是输出序列的第一个元素是4，输出的序列一定是4321。得到1342的过程如下：1入栈并出栈，得到部分输出序列1；2、3入栈，3出栈，得到部分输出序列13；4入栈并出栈，得到部分输出序列134；2出栈，得到最终结果1342。

答案：①

(5) 设输入序列为BASURN，则借助一个栈得不到的输出序列是(　　)。

① BUSANR　　② RNSABU　　③ URSANB　　④ ARUNSB

解答：输入序列为BASURN，输出序列不可能是RNSABU，理由是输出序列的第一个元素是R，一定是BASUR依次入栈，在输出序列中，U一定在S的前面。得到BUSANR的过程如下：B入栈，B出栈，得到部分输出序列B；A、S、U入栈并出栈，得到部分输出序列BUSA；R、N入栈并出栈，得到最终结果BUSANR。得到URSANB的过程如下：B、A、S、U入栈，U出栈，得到部分输出序列U；R入栈并出栈，得到部分输出序列UR；S、A出栈，得到部分输出序列URSA；N入栈并出栈，得到部分输出序列URSAN；B出栈，得到最终结果URSANB。得到ARUNSB的过程如下：B、A入栈，A出栈，得到部分输出序列A；S、U、R入栈，R、U出栈，得到部分输出序列ARU；N入栈并出栈，得到部分输出序列ARUN；S、B出栈，得到最终结果ARUNSB。

答案：②

(6) 在一个具有n个单元的顺序栈中，假设以地址低端作为栈底，以top作为栈顶指针，则当做退栈处理时，top变化为(　　)。

① top不变　　② top+=n　　③ top--　　④ top++

解答：由于是顺序栈，且以地址低端作为栈底，因此当做退栈处理时，top应指向当前栈顶元素的前一个元素，即top--。

答案：③

(7) 若循环队列的队头指针为 front,队尾指针为 rear,则队长的计算公式为(　　)。

① rear－front　　② front－rear　　③ rear－front＋1　④ 以上都不正确

解答：在具有 n 个单元的顺序存储的循环队列中,假设 front 和 rear 分别为队首指针和队尾指针,则循环队列长度的计算公式为(rear＋n－front)%n。

答案：④

(8) 栈和队列都是(　　)。

① 顺序存储的线性表

② 链式存储的线性表

③ 限制插入、删除位置的线性表

④ 限制插入、删除操作位置的非线性结构

解答：栈是一种特殊的线性表,它只允许在一端进行插入和删除操作。允许插入和删除的一端称为栈顶,另一端称为栈底。队列也是一种特殊的线性表,它只允许在一端进行插入操作,在另一端进行删除操作。允许插入的一端称为队尾,允许删除的一端称为队头,新插入的结点只能添加到队尾,要删除的结点只能是排在队头的结点。

答案：③

(9) 某线性表中最常用的操作是在最后一个元素之后插入一个元素及删除第一个元素,则采用(　　)存储方式最节省操作时间。

① 单链表　　② 队列　　③ 双链表　　④ 栈

解答：由题意可知,该线性表的插入操作在尾部,删除操作在头部,即先进先出,这正是队列的特点,所以采用队列存储方式最节省操作时间。

答案：②

(10) Equal("aaab","aaabc")的结果为(　　)。

① 1　　② 0　　③ －1　　④ 错误

解答：Equal()的功能是判断两个字符串是否相等,若相等则返回 1,否则返回 0。两个字符串相等的充要条件是两个字符串的长度相等,且各个对应位置上的字符都相同。"aaab"和"aaabc"的长度不相等,返回值为 0。

答案：②

2. 正误判断题

(1) 因为栈是一种线性表,所以线性表的所有操作都适用于栈。　　(　　)

解答：栈是一种特殊的线性表,它只允许在同一端进行插入和删除操作。

答案：×

(2) 队列是特殊的线性表,在队列的两端可以进行同样的操作。　　(　　)

解答：队列是一种特殊的线性表,它只允许在一端进行插入操作,在另一端进行删除操作。允许插入的一端称为队尾,允许删除的一端称为队头,新插入的结点只能添加到队尾,要删除的结点只能是排在队头的结点。

答案：×

(3) 如果两个串中含有相同的字符,则这两个串相等。　　(　　)

解答：若两个串的长度相等,并且各个对应位置上的字符都相同,则称这两个串相

等。若两个串中仅含有相同的字符，则这两个串不一定相等。例如："ab"和"ba"含有相同的字符，但这两个串不相等。

答案：×

(4) 栈是一种结构，既可以用顺序表表示，又可以用链表表示。 （ ）

解答：栈是一种特殊的线性表，它也有线性表的顺序存储和链式存储两种存储结构，分别称为顺序栈和链栈。

答案：√

(5) 队列这种结构是不允许在中间插入和删除数据的。 （ ）

解答：队列是一种特殊的线性表，它只允许在一端进行插入操作，在另一端进行删除操作，不允许在中间插入和删除数据。

答案：√

(6) 循环队列只能用顺序表实现。 （ ）

解答：使用循环队列的目的是避免顺序队列的假溢出现象，所以循环队列只能用顺序表实现。

答案：√

(7) 顺序栈的“栈满”与“上溢”是没有区别的，“栈空”与“下溢”是有区别的。 （ ）

解答：栈满时，若有元素入栈，则将产生上溢。栈空时，若进行出栈操作，则将产生下溢。由此可知，“栈满”与“上溢”有区别，“栈空”与“下溢”也有区别。

答案：×

(8) 顺序队列的“假溢出”现象是可以解决的。 （ ）

解答：顺序队列的“假溢出”现象可以利用循环队列或移动元素的方式解决。

答案：√

(9) 空串与空格串是一样的。 （ ）

解答：空串中没有字符，其长度为0；空格串中含有空格字符，其长度为空格的个数。

答案：×

(10) 链串的存储密度会影响到串的操作实现。 （ ）

解答：链串的结点越大，存储密度就越大，但这会给一些操作（如插入、删除、替换等）带来不便，可能会引起大量的字符移动。链串的结点越小（结点大小为1时），操作处理越方便，但存储密度会下降。

答案：√

3. 计算操作题

(1) 有一个字符串序列为3*y－a/y，写出利用栈把原序列改为字符串序列3y－*ay/的操作步骤。可用p表示扫描该字符串过程中顺序存取一个字符进栈的操作，用s表示从栈中取出一个字符加入新字符串尾的出栈操作。例如：将abc改为bca的操作步骤为ppspss。

解答：利用栈把字符串序列3*y－a/y改为字符串序列3y－*ay/的操作步骤为pspppspsspsppss。

(2) 设有编号为1、2、3、4、5的五辆列车顺序进入一个栈式结构的站台，已知最先开

出车站的前两辆车的编号依次为 3 和 4，请写出这五辆列车开出车站的所有可能的顺序。

解答：这五辆列车开出车站的顺序有 3 种：34521，34251，34215。

(3) 设 A 是一个栈，栈中的 n 个元素依次为 A_1，A_2，…，A_n，栈顶元素为 A_n。B 是一个循环队列，队列中的 n 个元素依次为 B_1，B_2，…，B_n，队头元素为 B_1。A 和 B 均采用顺序存储结构且存储空间足够大，现要将栈中的全部元素移到队列中，使得队列中的元素与栈中的元素交替排列，即 B 中的元素为 B_1，A_1，B_2，A_2，…，B_n，A_n。请写出具体的操作步骤，要求只允许使用一个辅助存储空间。

解答：具体操作步骤如下。

① 先将栈中的所有元素依次入队，此时栈空，队列为 B_1，B_2，…，B_n，A_n，A_{n-1}，…，A_1。

② 将 B_1，B_2，…，B_n依次出队、入队，此时队列为 A_n，A_{n-1}，…，A_1，B_1，B_2，…，B_n。

③ 将 A_n，A_{n-1}，…，A_1 出队、入栈，此时栈为 A_1，A_2，…，A_n（A_1 为栈顶），队列为 B_1，B_2，…，B_n。

④ 依次使 B_i出队、入队，A_i出栈、入队，此时队列为 B_1，A_1，B_2，A_2，…，B_n，A_n。

(4) 已知一个模式串为"abcaabca"，按照 KMP 算法求其 next[j]值。

解答：串"abcaabca"的 next[j]值为

j	0	1	2	3	4	5	6	7
模式	a	b	c	a	a	b	c	a
next[j]	−1	0	0	0	1	1	2	3

4. 算法设计题

(1) 写出 Ackermann 函数的递归算法。Ackermann 函数的定义为

$$\mathrm{Ack}(m,n)=\begin{cases} n+1 & m=0 \\ \mathrm{Ack}(m-1,1) & m\neq 0,n=0 \\ \mathrm{Ack}(m-1,\mathrm{Ack}(m,n-1)) & m\neq 0,n\neq 0 \end{cases}$$

解答：

```
int Ack(int m,int n)
{ if(m==0) return (n+1);
  else if(n==0) return(Ack(m-1,1));
       else return(Ack(m-1,Ack(m,n-1)));
}
```

(2) 已知压栈函数 Push(s,x)，弹栈函数 Pop(s,e)，初始化栈函数 InitStack(s)，判栈空函数 Empty(s)。编写算法，将任意一个十进制整数转换为任意（二至九）进制数并输出。

解答：利用辗转相除法将余数依次入栈，再依次出栈并输出。

```
void Conversion(int m, int n)
{ int e;sqstack S;
```

```
  initstack(&S);                    /*初始化空栈*/
  while(m!=0)
  { push(&S,m%n); m=m/n;}           /*余数进栈,更新被除数*/
    while(!empty(&S))
    { pop(&S,&e); printf("%d",e);   /*依次出栈并输出*/
  }
}
```

(3) 已知顺序栈的类型定义为

```
#define MAXSIZE 100                 /*栈初始空间大小*/
typedef int ElemType;
typedef struct
{ ElemType *base;                   /*栈底指针*/
  ElemType *top;                    /*栈顶指针*/
}qstack;                            /*栈的类型*/
```

请设计栈空和栈满的条件,并依此分别编写压栈和弹栈的算法。

解答：栈空的条件为S－＞top＝＝S－＞base;栈满的条件为S－＞top＝＝S－＞base＋MAXSIZE。

① 压栈算法。

```
int Push(qstack *S, ElemType x)
{ if(S->top==S->base+MAXSIZE)  return 0;  /*栈满*/
   *(S->top)=x;                     /*x存入栈顶位置*/
  S->top++;                         /*修改栈顶指针*/
  return 1;
}
```

② 弹栈算法。

```
int Pop(qstack *S, ElemType *e)
{ if(S->top==S->base) return 0;     /*栈空*/
  S->top--;                         /*修改栈顶指针*/
  *e=*(S->top);                     /*保存栈顶元素值*/
  return 1;
}
```

(4) 编写算法,用链栈实现带头结点的单链表逆置的操作。

解答：先初始化一个空的链栈,然后将单链表的结点依次入栈,最终的链栈就是单链表的逆置。

```
typedef int ElemType;
typedef struct node
{ ElemType data;                    /*数据域*/
  struct node *next;                /*指针域*/
}LinkStack,slink;                   /*链栈的结点类型与单链表的结点类型相同*/
```