第3章 栈和队列

CHAPTER

3.1 本章导学

1. 知识结构图

本章的知识结构如图 3-1 所示。

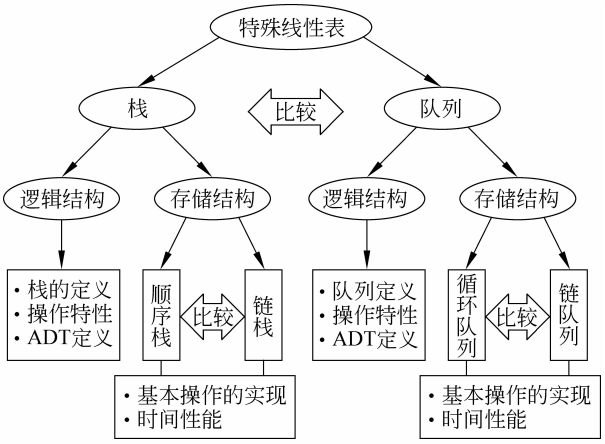

图 3-1 第 3 章知识结构图

2. 学习要点

本章的学习要从两条主线出发，一条主线是栈，另一条主线是队列，要以栈和队列的操作特性为切入点，并注意将栈和队列进行对比。

这部分有较多的经典应用，如汉诺塔问题、迷宫问题、八皇后问题、车厢重排问题、开关盒布线、舞伴问题等。要有意识地接触这些经典问题，深刻理解栈和队列在程序设计中的重要作用，在实践中培养数据结构应用能力和算法设计能力。

3. 重点整理

(1) 栈是限定仅在表尾进行插入和删除操作的线性表。栈中元素除了具有线性关系外，还具有后进先出的特性。

（2）栈的顺序存储结构称为顺序栈，顺序栈本质上是顺序表的简化。通常把数组中下标为0的一端作为栈底，同时附设指针top指示栈顶元素在数组中的位置。

（3）实现顺序栈基本操作的算法的时间复杂度均为$O(1)$。

（4）栈的链接存储结构称为链栈，通常用单链表表示，并且不用附加头结点。

（5）链栈的插入和删除操作只需处理栈顶即开始结点的情况，其时间复杂度均为$O(1)$。

（6）队列是只允许在一端进行插入操作，而另一端进行删除操作的线性表。队列中的元素除了具有线性关系外，还具有先进先出特性。

（7）顺序队列会出现假溢出问题，解决的办法是用首尾相接的顺序存储结构，称为循环队列。在循环队列中，凡是涉及队头或队尾指针的修改都需要将其求模。

（8）在循环队列中，队空的判定条件是：队头指针＝队尾指针；在浪费一个存储单元的情况下，队满的判定条件是：（队尾指针＋1）%数组长度＝队头指针。

（9）队列的链接存储结构称为链队列。链队列通常附设头结点，并设置队头指针指向头结点，队尾指针指向终端结点。

（10）链队列基本操作的实现本质上也是单链表操作的简化，插入只考虑在链队列的尾部进行，删除只考虑在链队列的头部进行，其时间复杂度均为$O(1)$。

3.2 重点难点释疑

3.2.1 浅析栈的操作特性

栈是限定仅在表尾进行插入和删除操作的线性表，栈中元素除了具有线性关系外，还具有后进先出的特性。需要强调的是，栈只是对线性表的插入和删除操作的位置进行了限制，并没有限定插入和删除操作进行的时间，也就是说，出栈可随时进行，只要某个元素位于栈顶就可以出栈。例如有三个元素a、b、c，按a、b、c的次序依次进栈，且每个元素只允许进一次栈，则可能的出栈序列有abc、acb、bac、bca、cba五种，设I代表入栈，O代表出栈，其操作示意图如图3-2所示。

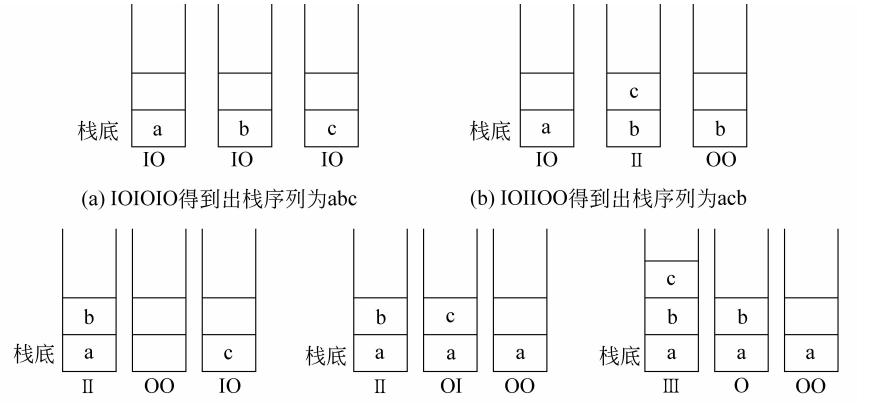

图3-2 栈操作示意图

3.2.2　递归算法转换为非递归算法

递归算法实际上是一种分而治之的方法，它把复杂问题分解为简单问题来求解。对于某些复杂问题(例如 hanio 塔问题)，递归算法是一种自然且合乎逻辑的解决问题的方式，但是递归算法的执行效率通常比较差。因此，在求解某些问题时，常采用递归算法来分析问题，用非递归算法来求解问题；另外，有些程序设计语言不支持递归，这就需要把递归算法转换为非递归算法。

将递归算法转换为非递归算法有两种方法，一种是直接求值，不需要回溯；另一种是不能直接求值，需要回溯。前者使用一些变量保存中间结果，称为直接转换法；后者使用栈保存中间结果，称为间接转换法，下面分别讨论这两种方法。

1. 直接转换法

直接转换法通常用来消除尾递归和单向递归，将递归结构用循环结构来替代。

尾递归是指在递归算法中，递归调用语句只有一个，而且是处在算法的最后。例如求阶乘的递归算法：

```
long fact(int n)
{
    if (n == 0) return 1;
    else return n * fact(n - 1);
}
```

当递归调用返回时，是返回到上一层递归调用的下一条语句，而这个返回位置正好是算法的结束处，所以，不必利用栈来保存返回信息。对于尾递归形式的递归算法，可以利用循环结构来替代。例如求阶乘的递归算法可以写成如下循环结构的非递归算法：

```
long fact(int n)
{
    int s = 1;
    for (int i = 1; i <= n; i++)
        s = s * i;                                    //用 s 保存中间结果
    return s;
}
```

单向递归是指递归算法中虽然有多处递归调用语句，但各递归调用语句的参数之间没有关系，并且这些递归调用语句都处在递归算法的最后。显然，尾递归是单向递归的特例。例如求斐波那契数列的递归算法如下：

```
int f(int n)
{
    if (n == 1||n == 0) return 1;
    else return f(n - 1) + f(n - 2);
```

```
}
```

对于单向递归，可以设置一些变量保存中间结果，将递归结构用循环结构来替代。例如求斐波那契数列的算法中用 s1 和 s2 保存中间的计算结果，非递归函数如下：

```
int f(int n)
{
    int i, s;
    int s1 = 1, s2 = 1;
    for (i = 3; i <= n; i++)
    {
        s = s1 + s2;
        s2 = s1;                          //保存 f(n-2)的值
        s1 = s;                           //保存 f(n-1)的值
    }
    return s;
}
```

2. 间接转换法

该方法使用栈保存中间结果，一般需根据递归函数在执行过程中栈的变化得到。其一般过程如下：

```
将初始状态 s0 进栈；
while（栈不为空）
{
    退栈，将栈顶元素赋给 s；
    if（s 是要找的结果）返回；
    else {
        寻找到 s 的相关状态 s1；
        将 s1 进栈；
    }
}
```

间接转换法在数据结构中有较多实例，如二叉树遍历算法的非递归实现、图的深度优先遍历算法的非递归实现等，请读者参考主教材中的相关内容。

3.2.3 循环队列中队空和队满的判定方法

对于循环队列有一个虽然小却十分重要的问题：如何确定队空和队满的判定条件。

如图 3-3(a)和图 3-3(c)所示，队列中只有一个元素，执行出队操作，则队头指针加 1 后与队尾指针相等(注意队头指针加 1 后要与数组长度求模)，即队空的条件是 front＝rear。

图 3-4(a)和图 3-4(c)所示数组中只有一个空闲空间，执行入队操作，则队尾指针加 1

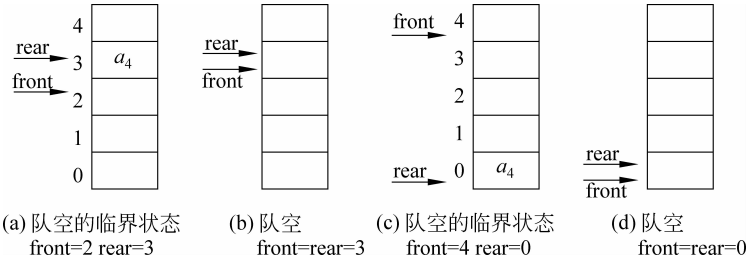

图 3-3　循环队列队空的判定

后与队头指针相等(注意队尾指针加 1 后要与数组长度求模),即队满的条件也是 front＝rear。

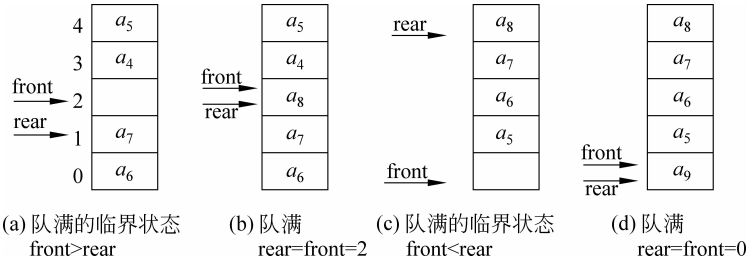

图 3-4　循环队列队满的判定

如何将队空和队满的判定条件区分开呢？有以下三种解决办法：

方法一：浪费一个数组单元,设存储循环队列的数组长度为 QueueSize,则队满的判定条件是：(rear＋1)mod QueueSize＝front,从而保证了 front＝rear 是队空的判定条件。具体算法请参见主教材。

方法二：设置一个标志 flag,当 front＝rear 且 flag＝0 时为队空,当 front＝rear 且 flag＝1 时为队满。相应地要修改循环队列类 CirQueue 的入队和出队算法。当有元素入队时,队列非空,所以将 flag 置 1,而当有元素出队时,队列不满,所以将 flag 置 0。

循环队列入队算法 Push

```
template <class T>
void CirQueue::Push(T x)
{
    if (front == rear && flag == 1) throw "overflow";
    flag = 1;
    rear = (rear + 1) % QueueSize;
    data[rear] = x;
}
```

循环队列出队算法 Pop

```
template <class T>
T CirQueue::Pop( )
{
    if (front == rear && flag == 0) throw "underflow";
    flag = 0;
    front = (front + 1) % QueueSize;
    return data[front];
}
```

方法三：设置一个计数器 count 来累计队列的长度，则当 count＝0 时队列为空，当 count＝ QueueSize 时队列为满，每入队一个元素 count 加 1，每出队一个元素 count 减 1。相应地要在循环队列类 CirQueue 中增加一个私有成员变量 count，并修改循环队列类 CirQueue 的入队和出队算法。

循环队列入队算法 Push

```
template <class T>
void CirQueue::Push(T x)
{
    if (count == QueueSize) throw "overflow";
    count ++ ;
    rear = (rear + 1) % QueueSize;
    data[rear] = x;
}
```

循环队列出队算法 Pop

```
template <class T>
T CirQueue::Pop( )
{
    if (count == 0) throw "underflow";
    count -- ;
    front = (front + 1) % QueueSize;
    return data[front];
}
```

3.3 习题解析

3.3.1 课后习题讲解

1. 填空题

(1) 设有一个空栈,栈顶指针为1000H,每个元素需要1个单位的存储空间,则执行push,push,pop,push,pop,push,push后,栈顶指针为(　　)。

【解答】 1003H。

(2) 栈结构通常采用的两种存储结构是(　　);其判定栈空的条件分别是(　　),判定栈满的条件分别是(　　)。

【解答】 顺序存储结构和链接存储结构(或顺序栈和链栈),栈顶指针top= －1和top=NULL,栈顶指针top等于数组的长度和内存无可用空间。

(3) (　　)可作为实现递归函数调用的一种数据结构。

【解答】 栈。

【分析】 递归函数的调用和返回正好符合后进先出性。

(4) 表达式a*(b+c)－d的后缀表达式是(　　)。

【解答】 abc+ * d－。

【分析】 将中辍表达式变为后缀表达式有一个技巧:将操作数依次写下来,再将算符插在它的两个操作数的后面。

(5) 栈和队列是两种特殊的线性表,栈的操作特性是(　　),队列的操作特性是(　　),栈和队列的主要区别在于(　　)。

【解答】 后进先出,先进先出,对插入和删除操作限定的位置不同。

(6) 循环队列的引入是为了克服(　　)。

【解答】 假溢出。

(7) 数组Q[n]用来表示一个循环队列,front为队头元素的前一个位置,rear为队尾元素的位置,计算队列中元素个数的公式为(　　)。

【解答】 (rear－front+n)% n。

【分析】 也可以是(rear－front)%n,但rear－front的结果可能是负整数,而对一个负整数求模,其结果在不同的编译器环境下可能会有所不同。

(8) 用循环链表表示的队列长度为n,若只设头指针,则出队和入队的时间复杂度分别是(　　)和(　　)。

【解答】 $O(1)$,$O(n)$。

【分析】 在带头指针的循环链表中,出队即是删除开始结点,这只需修改相应指针;入队即是在终端结点的后面插入一个结点,这需要从头指针开始查找终端结点的地址。

2. 单项选择题

(1) 一个栈的入栈序列是1,2,3,4,5,则栈的不可能的输出序列是(　　)。

A. 5,4,3,2,1　　B. 4,5,3,2,1　　C. 4,3,5,1,2　　D. 1,2,3,4,5

【解答】 C。

【分析】 此题有一个技巧：在输出序列中任意元素后面不能出现比该元素小并且是升序(指的是元素的序号)的两个元素。

(2) 若一个栈的输入序列是1,2,3,…,n,输出序列的第一个元素是n,则第i个输出元素是(　　)。

A. 不确定　　B. $n-i$　　C. $n-i-1$　　D. $n-i+1$

【解答】 D。

【分析】 此时,输出序列一定是输入序列的逆序。

(3) 若一个栈的输入序列是1,2,3,…,n,其输出序列是p_1，p_2，…，p_n,若$p_1=3$,则p_2的值(　　)。

A. 一定是2　　B. 一定是1　　C. 不可能是1　　D. 以上都不对

【解答】 C。

【分析】 由于$p_1=3$,说明1，2，3均入栈后3出栈,此时可能将当前栈顶元素2出栈,也可以继续执行入栈操作,因此p_2的值可能是2,但一定不能是1,因为1不是栈顶元素。

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

A. 顺序表　　B. 栈　　C. 队列　　D. 链表

【解答】 B。

【分析】 每个右括号与它前面的最后一个没有匹配的左括号配对,因此具有后进先出性。

(5) 在解决计算机主机与打印机之间速度不匹配问题时通常设置一个打印缓冲区,该缓冲区应该是一个(　　)结构。

A. 栈　　B. 队列　　C. 数组　　D. 线性表

【解答】 B。

【分析】 先进入打印缓冲区的文件先被打印,因此具有先进先出性。

(6) 一个队列的入队顺序是1,2,3,4,则队列的输出顺序是(　　)。

A. 4,3,2,1　　B. 1,2,3,4　　C. 1,4,3,2　　D. 3,2,4,1

【解答】 B。

【分析】 队列的入队顺序和出队顺序总是一致的。

(7) 栈和队列的主要区别在于(　　)。

A. 它们的逻辑结构不一样　　B. 它们的存储结构不一样

C. 所包含的运算不一样　　D. 插入、删除运算的限定不一样

【解答】 D。

【分析】 栈和队列的逻辑结构都是线性的,都有顺序存储和链接存储,有可能包含的运算不一样,但不是主要区别,任何数据结构在针对具体问题时包含的运算都可能不同。

(8) 设数组S[n]作为两个栈S1和S2的存储空间,对任何一个栈只有当S[n]全满时才不能进行进栈操作。为这两个栈分配空间的最佳方案是(　　)。

A. S1的栈底位置为0,S2的栈底位置为$n-1$

B. S1的栈底位置为0,S2的栈底位置为$n/2$

C. S1的栈底位置为0,S2的栈底位置为n

D. S1的栈底位置为0,S2的栈底位置为1

【解答】 A。

【分析】 两栈共享空间首先两个栈是相向增长的,栈底应该分别指向两个栈中的第一个元素的位置,并注意C++中的数组下标是从0开始的。

(9) 设栈S和队列Q的初始状态为空,元素$e1$,$e2$,$e3$,$e4$,$e5$,$e6$依次通过栈S,一个元素出栈后即进入队列Q,若6个元素出队的顺序是$e2$,$e4$,$e3$,$e6$,$e5$,$e1$,则栈S的容量至少应该是(　　)。

A. 6　　B. 4　　C. 3　　D. 2

【解答】 C。

【分析】 由于队列具有先进先出性,所以,此题中队列形同虚设,即出栈的顺序也是$e2$,$e4$,$e3$,$e6$,$e5$,$e1$。

3. 判断题

(1) 有n个元素依次进栈,则出栈序列有$(n-1)/2$种。

【解答】 错。

【分析】 应该有$\frac{(2n)!}{(n+1)(n!)^2}$种。

(2) 栈可以作为实现过程调用的一种数据结构。

【解答】 对。

【分析】 只要操作满足后进先出性,都可以采用栈作为辅助数据结构。

(3) 在栈满的情况下不能做进栈操作,否则将产生“上溢”。

【解答】 对。

(4) 在循环队列中,front指向队头元素的前一个位置,rear指向队尾元素的位置,则队满的条件是front=rear。

【解答】 错。

【分析】 这是队空的判定条件,在循环队列中要将队空和队满的判定条件区别开。

(5) 循环队列中至少有一个数组空间是空闲的。

【解答】 错。

【分析】 如果假定循环队列满足条件(rear+1)%Maxsize=front时为队满,则循环队列中至少有一个数组空间是空闲的。

4. 简答题

(1) 设有一个栈,元素进栈的次序为A,B,C,D,E,能否得到如下出栈序列,若能,请写出操作序列,若不能,请说明原因。

① C,E,A,B,D

② C,B,A,D,E

【解答】 ① 不能。因为在C、E出栈的情况下，A一定在栈中，而且在B的下面，不可能先于B出栈。

② 可以。设I为进栈操作，O为出栈操作，则其操作序列为IIIOOOIOIO。

（2）举例说明顺序队列的“假溢出”现象。

【解答】 假设有一个顺序队列，如图3-5所示，队尾指针rear=4，队头指针front=1，如果再有元素入队，就会产生“上溢”，此时的“上溢”又称为“假溢出”，因为队列并不是真的溢出了，存储队列的数组中还有2个存储单元空闲，其下标分别为0和1。

（3）在操作序列push(1)，push(2)，pop，push(5)，push(7)，pop，push(6)之后，栈顶元素和栈底元素分别是什么？

提示：push(k)表示整数k入栈，pop表示栈顶元素出栈。

【解答】 栈顶元素为6，栈底元素为1。其执行过程如图3-6所示。

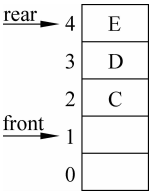

图3-5 顺序队列的假溢出

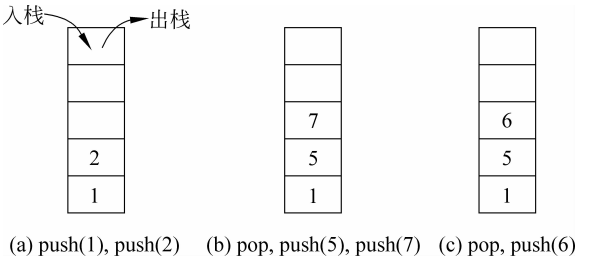

图3-6 栈的执行过程示意图

（4）在操作序列EnQueue(1)，EnQueue(3)，DeQueue，EnQueue(5)，EnQueue(7)，DeQueue，EnQueue(9)之后，队头元素和队尾元素分别是什么？

提示：EnQueue(k)表示整数k入队，DeQueue表示队头元素出队。

【解答】 队头元素为5，队尾元素为9。其执行过程如图3-7所示。

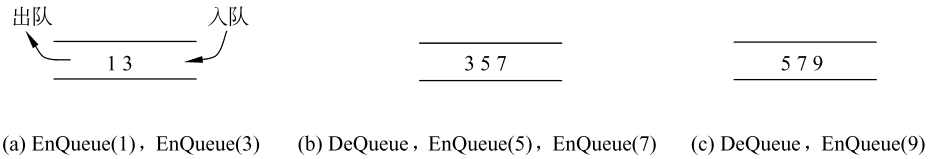

图3-7 队列的执行过程示意图

（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。下面序列中哪些是合法的？为什么？

① IOIIOIOO

② IOOIOIIO

③ IIIOIOIO

④ IIIOOIOO

【解答】 ①和④是合法序列，②和③是非法序列。