第 3 章　栈、队列和数组

3.1　考纲要求及分析

【考纲要求】

（一）栈和队列的基本概念
（二）栈和队列的顺序存储结构
（三）栈和队列的链式存储结构
（四）多维数组的存储
（五）特殊矩阵的压缩存储
（六）栈、队列和数组的应用

【考纲分析】

本章考查两个特殊的线性表——栈和队列，一个扩展的线性表——多维数组。本章要求：

（1）理解栈和队列的定义，运用栈和队列的定义与操作特性，解决计算机领域的实际应用问题，例如，子程序调用、表达式求值、括号匹配、打印机缓存等。

（2）评价顺序栈、链栈、共享栈、顺序队列、循环队列、链队列的存储方法，给出栈空、栈满、队空、队满的判定条件。

（3）掌握栈和队列的插入、删除、判空等基本操作的算法，并分析时空性能。

（4）理解数组的定义，掌握多维数组的存储方法和寻址方法。

（5）掌握特殊矩阵的压缩存储方法，能够对压缩存储的矩阵元素进行寻址计算。

【试题分析】

从历年考题情况看，本章基本每年都会有试题出现，出题形式主要以单项选择题为主，偶见综合应用题。主要出题点有栈和队列的操作特性、栈和队列的应用、循环队列、表达式求值、数组的存储与寻址、特殊矩阵的压缩存储等，有时会将栈和队列结合起来出题。本章内容比较简单，相对来说得分比较容易，因此本章的复习回报率较高，值得考生投入一定的精力进行复习。

3.2 栈


课件

3.2.1 考核知识点

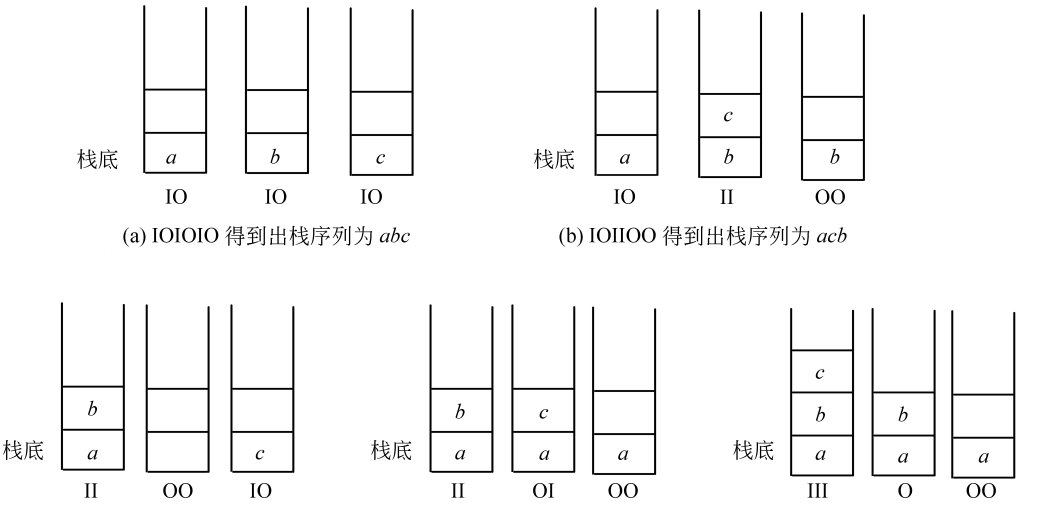
知识点名称	重要程度	难易程度	出题情况(年份用后两位表示)
栈的定义及操作特性	★★★★★	◆◆◇◇◇	单选：09-11、13、15、17-18、20
顺序栈	★★★☆☆	◆◆◇◇◇	
两栈共享空间	★☆☆☆☆	◆◆◆◇◇	
链栈	★★★☆☆	◆◆◇◇◇	
表达式求值	★★★★☆	◆◆◆◇◇	单选：12、14、18

3.2.2 重点难点释疑

栈的操作特性和表达式求值是本节两个出题频度较高的知识点，下面进行相关说明。

1. 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，操作示意图如图 3-1 所示。



(a) IOIOIO 得到出栈序列为 *abc*　(b) IOIIOO 得到出栈序列为 *acb*

(c) IIOOIO 得到出栈序列为 *bac*　(d) IIOIOO 得到出栈序列为 *bca*　(e) IIIOOO 得到出栈序列为 *cba*

图 3-1　栈的操作示意图

2. 表达式求值

表达式求值是编译程序的一个基本问题。设运算符有＋、－、＊、/、＃和圆括号，其中＃为表达式的定界符。中缀表达式的求值过程需要两个栈：运算对象栈 OPND 和运算符栈 OPTR，例如表达式(4 ＋ 2) ＊ 3－5 在求值过程中栈 OPND 和 OPTR 的状态如表 3-1 所示。求值算法的操作步骤为：

1. 栈 OPND 初始化为空，栈 OPTR 初始化为表达式的定界符＃。
2. 从左至右扫描表达式的每一个字符执行下述操作：
   2.1　若当前字符是运算对象，则入栈 OPND；
   2.2　若当前字符是运算符且优先级比栈 OPTR 的栈顶运算符的优先级高，则入栈 OPTR，处理下一个字符；
   2.3　若当前字符是运算符且优先级比栈 OPTR 的栈顶运算符的优先级低，则从栈 OPND 出栈两个运算对象，从栈 OPTR 出栈一个运算符进行运算，并将运算结果入栈 OPND，继续处理当前字符；
   2.4　若当前字符是运算符且优先级与栈 OPTR 的栈顶运算符的优先级相同，则将栈 OPTR 的栈顶运算符出栈，处理下一个字符。
3. 输出栈 OPND 中的栈顶元素，即表达式的运算结果。

表 3-1　表达式(4＋2)＊3－5 的求值过程

当前字符	运算对象栈 OPND	运算符栈 OPTR	说　明
		＃	初始化
(		＃，(	“(”的优先级比“＃”高，“(”入栈 OPTR
4	4	＃，(	4 入栈 OPND

续表

当前字符	运算对象栈 OPND	运算符栈 OPTR	说　明
+	4	#,(, +	“+”的优先级比“(”高,“+”入栈 OPTR
2	4,2	#,(, +	2 入栈 OPND
)	6	#, (	“)”的优先级比“+”低,2 和 4 出栈,“+”出栈,执行“+”运算并将结果 6 入栈 OPND
)	6	#	“)”的优先级与“(”相同,括号匹配,“(”出栈
*	6	#, *	“*”的优先级比“#”高,“*”入栈 OPTR
3	6,3	#, *	3 入栈 OPND
−	18	#	“−”的优先级比“*”低,3 和 6 出栈,“*”出栈,执行“*”运算并将结果 18 入栈 OPND
−	18	#,−	“−”的优先级比“#”高,“−”入栈 OPTR
5	18,5	#,−	5 入栈 OPND
#	13	#	“#”的优先级比“−”低,5 和 18 出栈,“−”出栈,执行“−”运算并将结果 13 入栈 OPND
#	13	#	“#”的优先级与“#”相同,求值结束,栈 OPND 的栈顶元素即为运算结果

3. 中缀表达式转换为后缀表达式

运算符在两个运算对象的中间称为中缀表达式,运算符在两个运算对象的后面称为后缀表达式,也称逆波兰式。例如,中缀表达式(4 + 2) * 3 − 5 对应的后缀表达式为 4 2 + 3 * 5 −。对算术表达式的后缀形式仅做一次扫描即可得到表达式的运算结果,无须考虑优先级和括号等因素,因此,很多编译程序在对表达式进行语法检查的同时,将其转换为对应的后缀形式。将一个中缀表达式转换为对应的后缀表达式只需用一个栈存放运算符,中缀表达式(4 + 2) * 3 − 5 转换为对应的后缀表达式的过程如表 3-2 所示。

表 3-2　中缀表达式转换为后缀表达式的过程

当前字符	后缀表达式	栈 OPTR	说　明
		#	初始化
(		#,(	“(”的优先级比“#”高,“(”入栈
4	4	#,(	输出 4
+	4	#, (, +	“+”的优先级比“(”高,“+”入栈
2	4 2	#, (, +	输出 2
)	4 2 +	#,(	“)”的优先级比“+”低,“+”出栈并输出
)	4 2 +	#	“)”的优先级与“(”相同,括号匹配,“(”出栈
*	4 2 +	#, *	“*”的优先级比“#”高,“*”入栈
3	4 2 + 3	#, *	输出 3

续表

当前字符	后缀表达式	栈 OPTR	说　明
－	4 2 ＋ 3 ＊	#	“－”的优先级比“＊”低，“＊”出栈并输出
－	4 2 ＋ 3 ＊	#，－	“－”的优先级比“#”高，“－”入栈
5	4 2 ＋ 3 ＊ 5	#，－	输出 5
#	4 2 ＋ 3 ＊ 5 －	#	“#”的优先级比“－”低，“－”出栈并输出
#	4 2 ＋ 3 ＊ 5 －	#	“#”的优先级与“#”相同，转换结束

3.2.3 典型题解析

一、单项选择题

【试题点拨】 主要考查栈的基本操作在不同存储结构下的执行过程，对于顺序栈注意栈底的位置和栈顶指针的变化，对于链栈注意如何用链表实现栈以及插入和删除操作的位置。栈的重要考点是元素以同样顺序进栈后判断出栈的不同情况，两栈共享空间也是一个可能的考点。

1. 经过以下栈运算后，变量 x 的值是(　　)。

```
InitStack(s); Push(s, a); Push(s, b); Pop(s, x); GetTop(s, x);
```

A. a　　B. b　　C. 1　　D. 0

答案 A

分析 本题要求熟悉栈的基本操作，理解所给运算的含义。InitStack(s)表示对栈 s 进行初始化；Push(s, a)表示将元素 a 压入栈 s 中；Pop(s, x)表示将栈 s 的栈顶元素弹出并送入变量 x 中；GetTop(s, x)表示取栈顶元素并送入变量 x 中但不删除该元素。

2. 经过以下栈运算后，StackEmpty(s)的值是(　　)。

```
InitStack(s); Push(s, a); Push(s, b); Pop(s, x); Pop(s, y);
```

A. a　　B. b　　C. 1　　D. 0

答案 C

分析 注意 StackEmpty(s)的返回值，判断栈空的操作定义是：如果栈 s 为空，则返回 1，否则返回 0。

3. 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}

答案 C

分析 此题有一个答题技巧: 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。

4. 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. $n-i+1$

答案 D

分析 此时,输出序列一定是输入序列的逆序。

5. 若一个栈的输入序列是1,2,3,…,n,其输出序列是$p_1,p_2,\cdots,p_n$,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

答案 C

分析 由于$p_1=3$,说明 1, 2, 3 依次入栈后 3 出栈,接下来可能将当前栈顶元素 2 出栈,也可能继续执行入栈操作,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。

6. 当字符序列 t3_依次通过栈,输出长度为 3 且可用作 C 语言标识符的序列有(　　)。

A. 4 个　　B. 5 个　　C. 3 个　　D. 6 个

答案 C

分析 输出长度为 3 说明字符序列全部出栈,可以作为 C 语言标识符的序列只能以字母 t 或下画线_开头,而栈的输出序列中以字母 t 或下画线_开头的有三个,分别是 t3_、t_3 和_3t。

7. 在一个具有n个单元的顺序栈中,假定以地址低端(即下标为0的单元)作为栈底,以 top 作为栈顶指针,当出栈时,top 的变化为(　　)。

A. 不变　　B. top=0;　　C. top=top-1;　　D. top=top+1;

答案 C

分析 栈底固定在数组的低端,出栈时栈顶指针 top 需要减 1;如果栈底固定在数组的高端,则出栈时栈顶指针需要加 1。

8. 向一个栈顶指针为 h 的带头结点的链栈中插入指针 s 所指的结点时，应执行(　　)。

A. h->next＝s;

B. s->next＝h;

C. s->next＝h; h->next＝s

D. s->next＝h->next; h->next＝s;

答案 D

分析 结点 s 应插在头结点的后面。

9. 从栈顶指针为top的链栈中删除一个结点(假设不带头结点)，用变量x保存被删除结点的值，则执行(　　)。

A. x＝top; top＝top->next;

B. x＝top->data;

C. top＝top->next; x＝top->data;

D. x＝top->data; top＝top->next;

答案 D

分析 删除链栈的第一个结点，即指针 top 指向的结点。选项 A 保存的是栈顶指针；选项 B 只保存了被删结点的值，没有执行删除操作；选项 C 保存的是被删结点的后继结点的值。

10. 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

答案 A

分析 在两栈共享空间中，两个栈是相向增长的，两个栈的栈底应该分别设置在数组的两端。注意，题目表明数组下标从 0 开始。

11. 对于表达式 A－B＊C/D＋E/F 扫描到 D 时，运算符栈 OPTR 的栈顶元素是(　　)。

A. ＊　　B. /　　C. # －＊　　D. # －/

答案 B

分析 在计算表达式的过程中，栈 OPND 和 OPTR 的变化过程如表 3-3 所示。

表 3-3 栈 OPND 和 OPTR 的变化过程

步骤	栈 OPND	栈 OPTR	算术表达式(下画线为当前扫描的字符)
初始		#	A－B＊C/D＋E/F#
1	A	#	A－B＊C/D＋E/F#
2	A	#－	－B＊C/D＋E/F#
3	AB	#－	B＊C/D＋E/F#
4	AB	#－＊	＊C/D＋E/F#
5	ABC	#－＊	C/D＋E/F#
6	AT_1(T_1＝B＊C)	#－/	/D＋E/F#
7	AT_1D	#－/	D＋E/F#

12. 表达式3＊2^(4＋2＊2－6＊3)－5在求值过程中当扫描到6时，对象栈和运算符栈为(　　)，其中^表示乘幂。

A. 3,2,4,2,2;# ＊ ^(＋ ＊ －　　B. 3,2,8;# ＊ ^(＋ ＊ －

C. 3,2,4,2,2;# ＊ ^(－　　D. 3,2,8;# ＊ ^(－

答案 D

分析 当扫描到 6 时，已经计算了 2＊2＝4 和 4＋4＝8，因此，对象栈为 3, 2, 8；运算符栈中还有未运算的运算符# ＊ ^(－。

13. 与中缀表达式 a＊b＋c/d－e 等价的后缀表达式是(　　)。

A. abcde＊/＋－　　B. ab＊cd/＋e－　　C. ab＊c＋d/e－　　D. abcd＊/＋e－

答案 B

分析 先计算 a＊b，后缀形式为 ab＊；再计算 c/d，后缀形式为 cd/；再将 ab＊和 cd/相加，后缀形式为 ab＊cd/＋；最后将 ab＊cd/＋与 e 相减，因此，后缀表达式为 ab＊cd/＋e－。

二、算法设计题

【试题点拨】 由于顺序栈和链栈基本操作的实现非常简单，因此，一般不会单独以算法设计题的形式出现，但树或图的相关内容(例如二叉树的遍历、图的深度优先遍历)用栈作为辅助数据结构，因此，要求能够熟练写出栈的操作语句。

1. 假设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。回答下列问题：

(1) 下面所示的序列中哪些是合法的？

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

(2) 通过对(1)的分析,设计算法判定所给的操作序列是否合法。若合法,返回 true,否则返回 false(假定被判定的操作序列已存入一维数组中)。

解答 在入栈出栈序列(仅由 I 和 O 组成)的任一位置,入栈次数(即 I 的个数)都必须大于或等于出栈次数(即 O 的个数),否则在形式上视为非法序列。由于题目中要求栈的初态和终态都为空,则整个序列的入栈次数必须等于出栈次数,否则在形式上视为非法序列。因此,A 和 D 是合法序列,B 和 C 是非法序列。算法如下:

```
int Judge(char A[ ])                          //判断字符数组 A 是否是合法序列
{
  int i, countI = 0, countO = 0;              //分别存储 I 和 O 的个数
  for (i = 0; A[i] != '\0'; i++)
  {
    if (A[i] == 'I') countI++;                //入栈次数增 1
    else {
      countO++;                               //出栈次数增 1
      if (countO > countI) return 0;          //出栈次数大于入栈次数
    }
  }
  if (countI != countO) return 0;
  else return 1;
}
```

【试题点拨】 在具有后进先出操作特性的问题中常采用栈作为辅助数据结构,由于栈的运算非常简单,在算法中可以直接写出有关栈的操作语句。

2. 假设一个算术表达式中可以包含三种括号:圆括号“(”和“)”,方括号“[”和“]”以及花括号“{”和“}”,且这三种括号可按任意的次序嵌套使用。编写算法判断给定表达式中所含括号是否配对出现。

解答 可以将给定表达式存入字符数组 A[n]中,然后依次考查每个字符,如果为左括号(包括“(”“[”“{”)则将其入栈,如果为右括号,则将其与栈顶元素比较,若匹配了一对括号,则将栈顶元素出栈,若不匹配,则退出算法返回不配对结果。对每个字符均考查完毕,若栈为空,则说明括号配对,否则说明缺少右括号或多余左括号。算法只考查左右括号,对其他字符不予处理。算法如下:

```
int Prool(char A[ ])
{
  char S[80];                                 //采用顺序栈,表达式最多 80 个字符
  int top = -1, i;                            //栈初始化
  for (i = 0; A[i] != '\0'; i++)
  {
```

```
    if (A[i] = '(' || A[i] = '[' || A[i] = '{')
      S[++top] = A[i];                          //将左括号入栈
    else {
      switch (A[i])
      {
        case ')': if (top == -1 || S[top--] != '(') return 0;
        case ']': if (top == -1 || S[top--] != '[') return 0;
        case '}': if (top == -1 || S[top--] != {') return 0;
      }
    }
  }
  if (top != -1) return 0;                      //栈不空,则括号不匹配
  else return 1;
}
```

3.3 队　列


课件

3.3.1 考核知识点

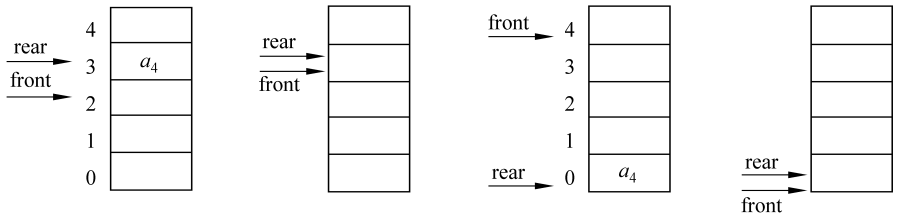
知识点名称	重要程度	难易程度	出题情况(年份用后两位表示)
队列的定义及操作特性	★★★★☆	◆◆◇◇◇	单选：09、10、18、21
顺序队列与循环队列	★★★★☆	◆◆◆◇◇	单选：11、14
链队列	★★★☆☆	◆◆◇◇◇	综合(应用)：19

3.3.2 重点难点释疑

对于循环队列有一个虽然小却十分重要的问题：确定队空和队满的判定条件。

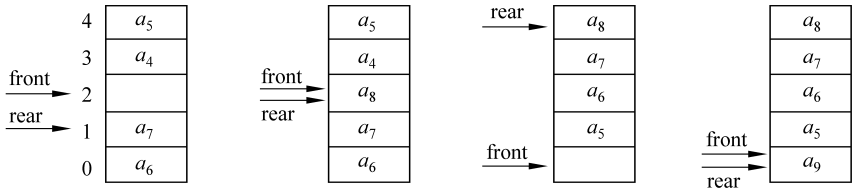
如图 3-2(a)和(c)所示，循环队列只有一个元素，执行出队操作，则队头指针加 1 后与队尾指针相等(队头指针加 1 后要与数组长度求模)，即队空的判定条件是 front = rear。

如图 3-3(a)和(c)所示，数组只有一个空闲空间，执行入队操作，则队尾指针加 1 后与队头指针相等(队尾指针加 1 后要与数组长度求模)，即队满的判定条件也是 front=rear。



(a) 队空的临界状态 1 rear > front　(b) 队空 front=rear=3　(c) 队空的临界状态 2 rear < front　(d) 队空 front=rear=0

图 3-2　循环队列队空的判定



(a) 队满的临界状态 1 front>rear　(b) 队满 rear=front=2　(c) 队满的临界状态 2 front<rear　(d) 队满 rear=front=0

图 3-3　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 时队列为空，当 front=rear 且 flag=1 时队列为满。当有元素入队时，队列非空，将 flag 置 1；当有元素出队时，队列不满，将 flag 置 0。

方法三：设置一个计数器 count 累计队列的长度，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。

3.3.3　典型题解析

一、单项选择题

【试题点拨】 主要考查队列的基本操作在不同存储结构下的执行过程，队列的重要考点是队列的操作特性、循环队列队空队满的判断、基于特性的应用等。

1. 一个队列的入队顺序是 1, 2, 3, 4，则队列的输出顺序是(　　)。

A. 4321　　B. 1234　　C. 1432　　D. 3241

答案　B

分析　队列的入队顺序和出队顺序总是一致的。

2. 循环队列存储在数组 A[0..m]中，则入队时的操作为(　　)。

A. rear＝rear＋1　　B. rear＝(rear＋1) mod (m－1)

C. rear＝(rear＋1) mod m　　D. rear＝(rear＋1) mod (m＋1)

答案 D

分析 注意，数组空间的长度为 m＋1，因此 rear＋1 的结果要与 m＋1 取模。

3. 若用一个长度为6的数组实现循环队列，且当前rear和front的值分别为0和3，则从队列中删除一个元素，再增加两个元素后，rear 和 front 的值分别为(　　)。

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

答案 B

分析 从循环队列中删除一个元素时，执行 front＝(front＋1) mod 6，向循环队列中插入一个元素时，执行 rear＝(rear＋1) mod 6。

4. 已知循环队列的存储空间为数组A[21]，front指向队头元素的前一个位置，rear指向队尾元素，假设当前 front 和 rear 的值分别为 8 和 3，则该队列的长度为(　　)。

A. 5　　B. 6　　C. 16　　D. 17

答案 C

分析 队列长度为(rear－front＋21) mod 21＝16，此时队列元素在数组中的位置依次是 A[9], A[10], …, A[20], A[0], A[1], …, A[3]。

5. 用不带头结点的单链表存储队列，队首指针指向队头结点，队尾指针指向队尾结点，则执行删除操作时，(　　)。

A. 仅修改队首指针

B. 队首指针和队尾指针都需要修改

C. 仅修改队尾指针

D. 队首指针和队尾指针可能都需要修改

答案 D

分析 链队列不带头结点，所以删除队头元素一定要修改队首指针，注意到队列长度为 1 的特殊情况，此时需要修改队尾指针。

6. 在链队列中，设指针 f 和 r 分别指向队首和队尾，则插入 s 所指结点的操作是(　　)。

A. f->next＝s; f＝s;　　B. r->next＝s; r＝s;

C. s->next＝r; r＝s;　　D. s->next＝f; f＝s;

答案 B

分析 结点 s 应插在队尾，因此，排除选项 A 和 D；s->next＝r 将结点 s 插在结点 r 的前面，因此，选项 C 错。

7. 设循环单链表表示的队列长度为 n，若只设头指针，则进队操作的时间性能为(　　)。

A. $O(n)$　　B. $O(1)$　　C. $O(n^2)$　　D. $O(n\log_2 n)$

答案 A

分析 进队操作需要查找尾结点，由于队列只设头指针，因此，必须从头指针开始遍历到表尾，然后将新结点插在尾结点的后面，时间性能为 $O(n)$。

8. 最不适合用作链队列的链表是(　　)。

A. 只带队首指针的双链表　　B. 只带队首指针的循环双链表

C. 只带队尾指针的循环双链表　　D. 只带队尾指针的循环单链表

答案 A

分析 在执行入队操作时需要修改队尾结点的指针域，由于双链表只带队首指针，查找队尾结点需要 $O(n)$的时间。对于选项 B、C、D，入队和出队的时间均为 $O(1)$。

二、算法设计题

【试题点拨】 链队列是用链接存储结构实现队列，有多种链表可以用来存储队列，例如同时带头指针和尾指针的单链表、带尾指针的循环单链表等，注意入队时元素是插在尾结点的后面，出队是在链表中删除第一个元素结点。

1. 假设以不带头结点的循环链表表示队列，并且只设一个指针指向队尾结点，但不设头指针。试设计相应的入队和出队算法。

解答 出队操作是在循环链表的头部进行，相当于删除开始结点，入队操作是在循环链表的尾部进行，相当于在终端结点之后插入一个结点。由于循环链表不带头结点，需要处理空表的特殊情况。算法如下：

```
void Enqueue(Node * rear, DataType x)
{
  Node * s = (Node * )malloc(sizeof(Node));  s->data = x;
  if (rear == NULL) {                        //处理空表的特殊情况
    rear = s;  rear->next = s;
  }
  else {                                     //处理除空表以外的一般情况
    s->next = rear->next;  rear->next = s;
    rear = s;
  }
```

```
}
DataType Dequeue(Node * rear)
{
  if (rear= =NULL) printf("队列为空");        //队列为空,无法执行出队操作
  else {
    Node * s = rear->next;
    if (s == rear) rear = NULL;               //链表中只有一个结点
    else rear->next = s->next;                //删除第一个结点
    free(s);
  }
}
```

【试题点拨】 在循环队列中,判断队空和队满的方法有多种,通常需要相应地修改入队和出队算法,有时还要修改循环队列的存储结构定义。

在循环队列中设置一个标志flag,当front＝rear且flag＝0时为队空,当front＝rear且flag＝1时为队满。设计相应的入队和出队算法。

解答 初始时队列为空,所以flag＝0;当有元素入队时,队列非空,将flag置1,当有元素出队时,队列不满,将flag置0。算法如下:

```
void EnQueue(CirQueue &Q, DataType x)        //flag为全局变量
{
  if (Q.front == Q.rear && flag == 1) printf("overflow");
  else {
    flag = 1;
    Q.rear = (Q.rear + 1) % QueueSize;
    Q.data[Q.rear] = x;
  }
}
DataType DeQueue(CirQueue &Q)
{
  if (Q.front == Q.rear && flag == 0) printf("underflow");
  else {
    flag = 0;
    Q.front = (Q.front + 1) % QueueSize;
    return Q.data[Q.front];
  }
}
```

如果设置一个计数器count累计队列的长度,则当count＝0时队列为空,当count＝QueueSize时队列为满。请设计相应的入队和出队算法。

解答 初始时队列为空,因此将count初始化为0。每入队一个元素将count加1,每出

队一个元素将 count 减 1。算法如下：

```
#define QueueSize 100
typedef struct                                  //定义循环队列存储结构
{
  DataType data[QueueSize];
  int count;                                    //计数器记载当前队列长度
  int front, rear;                              //队头、队尾指针
} CirQueue;
void EnQueue(CirQueue &Q, DataType x)
{
  if (Q.count == QueueSize) printf("overflow");
  else {
    Q.count++;
    Q.rear = (Q.rear + 1) % QueueSize;
    Q.data[Q.rear] = x;
  }
}
DataType DeQueue(CirQueue &Q)
{
  if (Q.count == 0) printf("underflow");
  else {
    Q.count--;
    Q.front = (Q.front + 1) % QueueSize;
    return Q.data[Q.front];
  }
}
```

一个双端队列 Q 是限定在线性表的两端都可以进行插入和删除操作的线性表，若采用顺序存储结构存储双端队列，写出在指定端 L(表示左端)和 R(表示右端)进行插入和删除操作的算法。

采用循环队列的存储思想，设 left 指向队列左端第一个元素(可以看成是左端的队头)的前一个位置，right 指向右端第一个元素(可以看成是右端的队头)的位置。则队列为空的条件是 left＝right，队列为满的条件是(right＋1) mod QueueSize＝left。在双端队列中执行插入操作需要指明操作的位置。算法如下：

```
enum OpertionTag {L, R};                        //L 表示队列的左端，R 表示队列的右端
#define QueueSize 100
typedef struct                                  //定义双端队列存储结构
{
  DataType data[QueueSize];
  int left, right;
```

```
} DulQueue;
void QueueInsert(DulQueue &Q, DataType x, OpertionTag side)
{
  if ((Q.right+1) % QueueSize == Q.left) {printf("上溢"); return; }
  if (side == L) {
  Q.data[Q.left] = x;                          //LEFT 已经指向插入的位置
  Q.left = (Q.left - 1) % QueueSize;           //左端指针在循环意义下减 1
  }
  else {
    Q.right = (Q.right + 1) % QueueSize;       //右端指针在循环意义下加 1
    Q.data[Q.right] = x;
  }
}
DataType QueueDelete(CirQueue &Q, OpertionTag side)
{
  if (Q.left == Q.right) {printf("下溢"); return; }
  if (side == L){
    Q.left = (Q.left + 1) % QueueSize;         //左端指针在循环意义下加 1
    return Q.data[Q.left];                     //左端指针指向左端被删除元素
  }
  else {
    DataType x = Q.data[Q.right];              //右端指针已指向右端被删除元素
    Q.right = (Q.right - 1) % QueueSize;       //右端指针在循环意义下减 1
    return x;
  }
}
```

3.4 多维数组的存储

课件

3.4.1 考核知识点

知识点名称	重要程度	难易程度	出题情况(年份用后两位表示)
数组的定义	★☆☆☆☆	◆◇◇◇◇	
数组的存储与寻址	★★★☆☆	◆◆◇◇◇	单选：21 综合(应用)：11

3.4.2 典型题解析

单项选择题

【试题点拨】 本节的题目比较简单，主要考查数组的定义、数组在行优先或列优先存储下数组元素的寻址等。

将数组称为随机存取结构是因为(　　)。

A. 数组元素是随机的

B. 对数组任一元素的存取时间是相等的

C. 随时可以对数组进行访问

D. 数组的存储结构是不确定的

B

存取结构是在一个数据结构上对查找操作的时间性能的一种描述，随机存取是指按位置查找的时间复杂度是 $O(1)$。

下面的说法中不正确的是(　　)。

A. 数组是一种线性结构

B. 数组是一种定长的线性结构

C. 除了插入与删除操作外，数组的基本操作还有存取、修改、检索和排序等

D. 数组的基本操作有存取、修改、检索和排序等，没有插入与删除操作

C

多维数组属于线性表的推广，数组被创建以后，其维数和每维中的元素个数就是确定的，所以数组通常没有插入和删除操作。

如果一维数组a[50]和二维数组b[10][5]具有相同的基类型和首地址，那么在以列优先方式存储时，a[18]的地址和(　　)的地址相同。

A. b[1][7]　　B. b[1][8]　　C. b[8][1]　　D. b[7][1]

C

a[18]是数组 a 的第 19 个元素，b[8][1]＝1×10＋9＝19，即 b[8][1]是数组 b 按列优先存储的第 19 个元素。

二维数组A中的每个元素均是由6个字符组成的串，行下标的范围为0～8，列下标的范围为 0～9，若二维数组 A 按行优先方式存储，则元素 A[8][5]的起始地址与二维数组 A 按列优先方式存储时元素(　　)的起始地址一致。

A. A[8][5]　　B. A[3][10]　　C. A[5][8]　　D. A[4][9]

答案　D

分析　元素 A[8][5]按行优先存储的起始地址为 $d+8\times10+5=d+85$。设元素 A[i][j]按列优先存储的起始地址与之相同，则 $d+j\times9+i=d+85$，解此方程得 $i=4$，$j=9$。

5.　三维数组 A[8, 8, 10]采用行优先方式从地址 d 开始存放，假设每个元素占用的存储空间大小均为 l，则元素 A[3, 2, 8]的存储位置是(　　)。

A. $d+198\times l$　　B. $d+108\times l$　　C. $d+268\times l$　　D. $d+13\times l$

答案　C

分析　LOC(A[3, 2, 8])=LOC(A[0, 0, 0])+(3×8×10+2×10+8)×l=$d+268\times l$。

3.5　矩阵的压缩存储

课件

3.5.1　考核知识点

知识点名称	重要程度	难易程度	出题情况(年份用后两位表示)
对称矩阵的压缩存储	★★★☆☆	◆◆◇◇◇	单选：18、20
三角矩阵的压缩存储	★★☆☆☆	◆◆◇◇◇	
对角矩阵的压缩存储	★★★☆☆	◆◆◆◇◇	单选：16
稀疏矩阵的压缩存储	★★★☆☆	◆◆◆◇◇	单选：17

3.5.2　重点难点释疑

在特殊矩阵的压缩存储中，矩阵元素的寻址计算不要死记公式，要掌握压缩存储方法和寻址方法，这样才能灵活处理。无论是按行优先还是按列优先，特殊矩阵压缩存储的基本思想都是一致的，需要注意矩阵中行下标和列下标的范围，以及存储矩阵的一维数组的起始下标。

例1 下三角矩阵A的行下标和列下标的范围为$1\sim n$和$1\sim m$，将下三角元素按行优先存储到数组SA[$n(n+1)/2$]中，则元素$a_{ij}(i\geqslant j)$存储到SA[k]，k的值是多少？

解答 下三角元素按行优先方式存储，元素$a_{ij}(i\geqslant j)$存储位置的计算过程如下。

(1) 计算第1行至第$i-1$行存储的元素个数：$1+2+3+\cdots+i-1=i(i-1)/2$。

(2) 第i行存储元素$a_{i1}\sim a_{ij}$，计算元素个数：j。

(3) 计算存储元素总数：$i(i-1)/2+j$。

(4) 数组SA下标从0开始，因此$k=i\times(i-1)/2+j-1$。

例2 将例1中下三角矩阵A的行下标和列下标的范围改为$0\sim n-1$和$0\sim m-1$，则k的值是多少？

解答 下三角元素按行优先方式存储，元素$a_{ij}(i\geqslant j)$存储位置的计算过程如下。

(1) 计算第0行至第$i-1$行存储的元素个数：$1+2+3+\cdots+i=i(i+1)/2$。

(2) 第i行存储元素$a_{i0}\sim a_{ij}$，计算元素个数：$j+1$。

(3) 计算存储元素总数：$i(i+1)/2+j+1$。

(4) 数组SA下标从0开始，因此$k=i\times(i+1)/2+j$。

例3 将例1中的按行优先存储改为按列优先存储，则k的值是多少？

解答 下三角元素按列优先方式存储，元素$a_{ij}(i\geqslant j)$存储位置的计算过程如下。

(1) 计算第1列至第$j-1$列存储的元素个数：$n+(n-1)+\cdots+(n-j+2)$。

(2) 第j列存储元素$a_{jj}\sim a_{ij}$，计算元素个数：$i-j+1$。

(3) 计算存储元素总数：$n+(n-1)+\cdots+(n-j+2)+(i-j+1)$。

(4) 数组SA下标从0开始，因此$k=(j-1)\times(2n-j+2)/2+(i-j)$。

3.5.3 典型题解析

一、单项选择题

【试题点拨】 主要考查特殊矩阵压缩存储后矩阵元素的寻址。注意矩阵元素的下标范围以及存储数组的起始下标。

1. 对特殊矩阵采用压缩存储的目的主要是(　　)。

A. 使表达变得简单　　B. 使矩阵元素的存取变得简单

C. 去掉矩阵中的多余元素　　D. 减少不必要的存储空间

答案 D

分析 在特殊矩阵中，有很多值相同的元素且分布有规律，可以为值相同的元素分配一个存储空间。

2. 设A[N][N]是对称矩阵，将下三角（包括对角线）按行优先存储到一维数组T[N(N+1)/2]中，则上三角元素A[i][j]对应T[k]的下标k是（　　）。

A. $j(j+1)/2+i-1$　B. $j(j-1)/2+i-1$　C. $j(j+1)/2+i$　D. $j(j-1)/2+i$

答案 C

分析 上三角元素A[i][j]的地址即是其对称元素A[j][i]的地址，注意到数组下标均从0开始，则 $k=(1+2+\cdots+j)+i=j(j+1)/2+i$。

3. 有一个n行n列的对称矩阵$\boldsymbol{A}$，将其下三角部分按行优先方式存放在一维数组B中，A[0][0]存放于B[0]中，则第i行的对角元素A[i][i]存放于B中（　　）处。

A. $(i+3)\times i/2$　B. $(i+1)\times i/2$

C. $(2n-i+1)\times i/2$　D. $(2n-i-1)\times i/2$

答案 A

分析 设A[i][i]是压缩存储后的第k个元素，第0行存储1个元素，第1行存储2个元素，以此类推，第$i-1$行存储i个元素，A[i][i]在第i行是第$i+1$个元素，则 $k=(1+2+\cdots+i)+i+1=i\times(i+3)/2+1$，注意到数组B的下标从0开始，则元素A[i][i]在B中的下标是 $i\times(i+3)/2$。

4. 若将n阶上三角矩阵$\boldsymbol{A}$按列优先方式压缩存储在一维数组B[$n(n+1)/2$]中，则存放到B[k]中的非零元素a_{ij}（$1\leqslant i, j\leqslant n$）的下标$i$, j与k的对应关系是（　　）。

A. $\frac{i(i+1)}{2}+j$　B. $\frac{i(i-1)}{2}+j-1$　C. $\frac{j(j+1)}{2}+i$　D. $\frac{j(j-1)}{2}+i-1$

答案 D

分析 由于按列优先方式，因此元素a_{ij}的左面有$j-1$列，共计 $1+2+\cdots+(j-1)=j(j-1)/2$ 个元素，元素a_{ij}在第j列是第i个元素，注意到数组B的下标从0开始，则 $k=\frac{j(j-1)}{2}+i-1$。

5. 若将n阶下三角矩阵$\boldsymbol{A}$按列优先方式压缩存放在一维数组B[1..$n(n+1)/2$]中，则存放到B[k]中的非零元素a_{ij}（$1\leqslant i, j\leqslant n$）的下标$i$, j与k的对应关系是（　　）。

A. $\frac{(j-1)(2n-j+1)}{2}+i-j$　B. $\frac{(j-1)(2n-j+2)}{2}+i-j+1$

C. $\frac{(j-1)(2n-j+2)}{2}+i-j$　D. $\frac{(j-1)(2n-j+2)}{2}+i-j-1$